Smart Contracts for Building Access Control

by

Leepakshi Bindra

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Large commercial buildings are complex cyber-physical systems containing expensive and critical equipment that ensure the safety and comfort of their numerous occupants. Yet occupant and visitor access to spaces and equipment within these buildings are still managed through unsystematic, inefficient, and human-intensive processes. As a standard practice, long-term building occupants are given access privileges to rooms and equipment based on their organizational roles, while visitors have to be escorted by their hosts.

Existing methods use a centralized infrastructure to delegate access to occupants, and sometimes visitors. A centralized technique is dependent on a single authority and requires the authority to be online all the time. Also, this technique is very vulnerable to failure as the whole system is compromised if the central server is attacked. These shortcomings make the traditional approach conservative and inflexible.

In this thesis, we describe a methodology that can flexibly and securely manage building access privileges for long-term occupants and short-term visitors alike, taking into account the risk associated with accessing each space within the building. Our methodology relies on blockchain smart contracts to describe, grant, audit, and revoke fine-grained permissions for building occupants and visitors, in a decentralized fashion. Access for visitors is described through smart contracts that use information of the event time, destination location and privilege of the individuals. The accessible spaces are specified through a process that leverages the information compiled from Brick and

BOT models of the building. BOT representation of the building help plan the spaces to which access should be provided to the visitor. To compute the risk of permitting an individual to enter a space, Brick models are employed to calculate sensitivity and security information of all the spaces.

We illustrate the proposed method through a typical application scenario in the context of a real office building and argue that it can greatly reduce the administration overhead, while, at the same time, providing fine-grained, auditable access control. We design and implement a commercial building simulator that imitates presence and movement of people in the building. The data synthesized by the simulator is used to evaluate the performance of the proposed system. We measure the delay in performing various access management tasks to assess the efficiency of the access management system. Scalability of the proposed solution is determined by calculating throughput and latency of the transactions on the blockchain network. The evaluation concludes that the most common type of requests made to the access management system is to verify access for individuals to a space, which take 0.26 and 0.37 seconds with different loads and computation power, which is typically within the acceptable range in real applications.

Preface

This thesis is an original work by Leepakshi Bindra. Segments from this thesis has been published in the following literature:

- Bindra, L., Eng, K., Ou, Y., Ardakanian, O. and Stroulia, E., 2019, November. Indoor Path Planning and Decentralized Access Control in Commercial Buildings. In Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (pp. 385-386).

- Bindra, L., Lin, C., Stroulia, E. and Ardakanian, O., 2019, May. Decentralized access control for smart buildings using metadata and smart contracts. In 2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS) (pp. 32-38). IEEE.

- Bindra, L., Eng, K., Ou, Y., Ardakanian, O., Stroulia, E.: Flexible, Decentralized Access Control for Smart Buildings with Smart Contracts. (Under review) ACM Transactions on Cyber-Physical Systems.

Acknowledgements

First and foremost, I would like to thank my advisors, Eleni Stroulia and Omid Ardakanian. It has been a privilege to work under such kind, thoughtful, and enthusiastic professors, who are always available with constructive and helpful feedback.

I would like to extend my sincere thanks to the members for serving on my defense committee.

I would like to appreciate and thank Yancheng Ou and Kalvin Eng for all the help they have provided.

Thank you.

Leepakshi Bindra

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Applications of Internet of Things (IoT) are fast increasing in industrial, commercial and domestic environments. According to a study conducted by IHS Markit, the number of connected IoT devices is likely to grow to 125 billion in 2030 [27]. Transportation, supply chain, retail, manufacturing and healthcare are some of the many industries that utilize IoT devices and technology for information transfer and analytics. Using sensors embedded in the vehicles, or mobile devices, it is possible to offer services like optimized route suggestions, collision prevention, and autonomous driving. In supply chain management, IoT helps make connections between supply chain entities and processes to track movement of products at each stage, providing complete information during the entire life cycle of products. IoT provides accurate real-time visibility into flow of materials for retail manufacturing, warehousing and retail delivery. IoT technology has also been widely utilized to complement and strengthen existing healthcare services by interconnecting various medical resources.

1.1 IoT in the Built Environment

Modern commercial buildings are complex cyber-physical systems. They are increasingly being equipped with sensors and actuators, ranging from surveillance cameras and card readers for security and access control, to thermostats and air-quality sensors feeding into the Heating, Ventilation, and Air Conditioning (HVAC) system, which controls the indoor environment while maintaining occupants comfort [18, 6]. These buildings represent substantial financial investments and the management of their security is even more critical compared to older buildings.

The vast network of interconnected devices can monitor the surroundings and control the physical world. Sensors are used in various devices like thermostats, lighting systems, smart blinds and security systems in the built environment. These devices use sensor data in control loops to adjust ambient temperature and lighting to meet occupant comfort requirements. The information gathered by smart devices can be analyzed to understand occupant presence and actions in the built environment.

The usage of IoT devices requires a lot of considerations for various challenges that they bring. The common challenges include security and privacy. Physical objects, such as smart door locks or lamps in a smart building, are being integrated into the Internet with networking and processing abilities. This allows services and resources to be accessible via mobile devices anytime and anywhere, making objects vulnerable to attacks and the collected data to intrusive inferences. There have been tremendous efforts in recent years to address security issues in the IoT. Some of these approaches target security issues at a specific layer, whereas, other approaches aim at providing end-to-end security for IoT [32]. Blockchain technology is now being adopted by the industry and research community as a technology that could play a major role in managing, controlling IoT devices and protecting the integrity of user data in a decentralized fashion [21, 42, 63].

Another challenge is user's access control in smart buildings. Traditional methods depend on a central third party to manage access control to spaces in buildings. A central authority once attacked compromises the whole access management system, making this approach vulnerable. Secondly, long-established methods of access delegation use swipe cards. For new or temporary visitors, temporary access management becomes tedious for building manager. A common protocol is that a visitor is escorted by someone (a long-term occupant or a security personnel) with an access card at all times. This issue of access control needs to be tackled in smart commercial buildings as well as smart homes.

Lastly, representation of IoT devices in a commercial building is done through “labels” that represent the function, type, location and relationships of the device in an abrupt naming convention. These are accessed through the Building Management System (BMS) and lack consistency between buildings. Hence, the challenge to use the BMS data for smart building application development arises. This can be tackled by using a semantic data model that uses uniform vocabulary to represent the devices, their function and relationships [7].

1.2 The Problem: Access Management in Commercial Buildings

Physical access control restricts access to physical spaces in a building, for example, controlling who can access which parts of a commercial building or how users can move within regulated spaces such as airports or hospitals. As physical spaces are usually comprised of subspaces, such as rooms connected by doors, a visitor needs to follow a path to arrive at the destined room, which requires that the visitor should have access to each door on the path.

Commercial buildings are difficult to navigate for any new visitor, and can be overwhelming if the person has to reach a particular room in the least amount of time. For instance, Bob is a new employee in an organization and has to reach a meeting room in his new office building. In order to manage his time well, he would like to know the way from the entrance of the building to the meeting room. He may also need to pass doors in the hallway that require every employee to scan their card and thus, would require access to such spaces a prior. Along with these, he should also be careful of not entering sensitive and restricted spaces. Access-control requirements for physical spaces are required to express constraints on the access paths through space and time. In the example above, a requirement might be that Bob should be able to access the office from 8am to 8pm from Monday through Friday until he leaves the organization. Another very useful scenario where such path planning and access control is of utmost use is a hospital. Visitors in hospitals are in distress and finding their way to the right location in the hospital building in case of emergencies is an additional burden. The solution to avoid getting lost and ending up in prohibited areas is path planning through the building. Building ontologies like BOT [46] and BIM [25] that describe the physical structure of the building can be employed for path planning. Determining the accessibility and sensitivity of different spaces in the building can be achieved with another ontology Brick[7], which defines functional elements and relationships in the building. This is further discussed in Chapter 4.

Reasoning about and managing access to these buildings require different access policies for different types of users. For example, as a standard practice, long-term building occupants such as employees who work in the building are given access privileges to their offices and shared spaces, based on their organizational roles; on the other hand, facilities-management personnel typically have access to the more restricted spaces where equipment is installed and also to the Building Management System (BMS) which enables them to monitor and control the equipment settings. Occupants may access the sensing and control devices, such as light switches and thermostats, in the spaces to which they have physical access even though these devices may impact building areas beyond the room in which they are physically located; for example, a thermostat located in a room can determine the temperature setpoint of multiple adjacent rooms. Finally, visitors tend to have limited access, and are frequently required to be escorted by building occupants to the meeting rooms where their business is taking place; they might control equipment in these rooms, but only for the duration of their meeting. In large commercial buildings, this approach implies substantial administrative overhead and exposes the building infrastructure to various security threats.

Many existing authorization systems [62, 41, 49] rely on a trusted central authority. However, if an attacker compromises this system, they can subvert the authorization policy of the entire system which poses a fundamental threat. The existing authorization method LDAP [62] uses Role-Based Access Control (RBAC) with a single central authority.

The work in [52] proposes another generic authorization Framework for the Internet-of-Things. It supports fine-grained and flexible access control for any constrained object with low power and memory resources. the methodology is based on current Internet standards and access control solutions such as XACML and Security Assertion Markup Language (SAML). However, it also introduces a central third party authorization engine to handle access control.

Using smart contracts for authorization and delegation of trust was originally proposed in WAVE [3]. WAVE is an IoT identity management, authentication, and authorization service defined on the public Ethereum blockchain. WAVE uses smart contracts deployed on the permissionless blockchain as a global ledger for all authorizations, Delegation of Trust (DoT), and revocations, guaranteeing that all participants know the current state of all permissions. BOSSWAVE [4] builds on WAVE to provide democratized access to the physical resources in buildings. It explored the usage of blockchain to enhance access control in building operating systems. A.Ouaddah et al. [44] used blockchain to store and audit access control policies.

1.3 Smart Contracts for Access Management based on Sensitivity of Indoor Paths

Ideally, an automated solution is needed to efficiently manage the access privileges of a building's occupants. Traditional role-based access-control models adopted in existing access control systems are unwieldy, in that they require the specification of all roles and their relative authority, which is a challenge in large buildings occupied by multiple organizations, each one with their own different role hierarchy. The proposed methodology manages building access control inspired by the concept of *airline boarding-passes* and the workflows around them. At any point in time, the airport is used by numerous airlines that manage their own flights at their corresponding gates and are responsible for issuing boarding passes to their passengers. Boarding passes become available shortly before passengers travel, enabling passengers to go through security, access their gate, and board their plane at the right time. Each boarding pass is associated with an individual traveler, and is valid only during a short period before the flight departure. During this period, security personnel are able to scan and verify the boarding pass. This methodology is envisioned to be implemented independent of any pre-existing access control delegation strategies, but still be co-existent with already implemented access control systems.

This methodology can flexibly and securely manage building access privileges for long-term occupants and short-term visitors alike, addressing the challenges and risks mentioned above. A set of services for *fine-grained* decentralized management of people's access privileges within a commercial building are established. The term fine-grained means (a) person-centric instead of role-based, (b) tailored to different space/system granularities, and (c) spanning multiple timescales. The underlying intuition for the work presented in this thesis is that *if a person is authorized to have physical access to a particular location in a building, then they also have access and opportunity to manipulate* the sensors and control points in this space. This is because, in most cases, there is no additional access-control beyond placing the equipment behind a locked door. Also by implication, *if a person should not have access to some control points, they should not be authorized to access the space where this equipment is located.*

The methodology relies on blockchain smart contracts to describe, grant, audit, and revoke fine-grained permissions for building occupants and visitors in a decentralized fashion [4]. The smart contracts are specified through a process that leverages the information compiled from the BOT [46] and Brick [7] models of the building's spatial structure, equipment, and their relations. These models use Resource Description Framework (RDF), which is a general-purpose language for representing information as a graph. RDF uses a textual syntax called Turtle [9] in which RDF graphs are written in a compact and natural text form.

The information from the building models enables our methodology to grant an individual with just the right access privileges to let them reach their destination within the building. This intuition is implemented in the form of smart contracts, through which space-access privileges are given to (and revoked from) individuals using an API that can be invoked by different software applications. The underlying *access-control service* is responsible for accepting or rejecting individual access requests based on the currently valid smart contracts and their implications regarding access to sensors and actuators. We illustrate the proposed method through a typical use case in the context of a real building and we argue that it can greatly reduce the administration overhead, while at the same time providing fine-grained, auditable access control.

Simulations enable researchers to synthesize focused replications of important activities and events under study. Rational simulations help generate a lot of test data for evaluating various systems in the same environment. Synthesized environments are easily modifiable and refinable allowing researchers to experiment, analyze and fine-tune their models and associated algorithms efficiently. Hence, a system designed on simulation studies would most likely to be a robust and inclusive design. Also, a simulation model that mimics an existing real world space is most likely to generate more data about the environment and answer more questions than the target actual space. This capability is an essential tool to use in establishment and deployment of smart city projects [26].

To evaluate and show the usefulness of the proposed solution, a simulator has been built to work on top of the services that provide path information and manage access. Simulating the presence and movement of people in a commercial building facilitates measuring the throughput of the proposed access management solution. The proposed solution should be able to work under high load, which is also assessed using the simulator.

The simulator is designed to synthesize movement of n number of people in a building for a duration of d days. The simulator creates a set of meetings during a day for a specific number of visitors and building occupants and then generates calendar files for each meeting which include meeting times, locations and participant information. The information from these calendar files is used to manage permissions for visitors and building occupants using our proposed service, the access control service, which runs on top of blockchain to make transaction requests. The set of accessible spaces and subspaces for each participant are extracted from building ontologies as described in Section 4.2. This helps illustrate a typical use case in the context of a real building which shows that the proposed solution can greatly reduce the administration overhead, while at the same time providing fine-grained, auditable access control.

An agent is formulated for each host and visitor to delegate access to spaces in the building and to verify access to the spaces when the visitor tries to enter them. The agents and the simulator work along with the services and the user interface to perform access delegation, path planning and system evaluation. The simulator utilizes stochastic processes to generate a meeting schedule and returns calendar files for each meeting to the front end application called the **Calendar Application** for smart buildings.

This application parses an *iCalendar* (ics) file the meeting host uploads, extracts the meeting information for each meeting, like the date, start($\mathrm{t_s}$) and end ($\mathrm{t_e}$) time of the meeting, the host and attendee email IDs and meeting room number ($\mathrm{r_d}$). The application queries the path planning service, which uses BOT and Brick to find paths between two locations (Section 4.2), to get the paths and their sensitivity costs for all paths from entrances of the building to the destination room. The meeting host (agent) is prompted to select a path for each meeting, depending on the cost of the paths. Along with this, the Calendar application also sends requests to the access control service to check if the meeting participants are known to the system. The selected path information is passed onto the access control services along with the request for creation of access rules for each participant. In the real-time scenario, a unique access QR code is generated for each participant which encodes the event unique ID extracted from the ics file, the host identity, and the participant identity, which is emailed to the meeting participant. When a meeting participant arrives in the building and scans their QR code at a door on the selected pathway, the smart lock interacts with the calendar application, which further checks the time-restricted access permission with the access-control service. However, for evaluation, the navigation of visitors is synthesized by the simulator, which calls the access control service to check if a user has access to a space at a given time. The details of the interaction between different services and components is discussed later in Chapter 4.

1.4 Contributions

The contributions of this thesis are:

- We develop a smart-contract based solution to flexibly manage access control for large commercial buildings. Distributed services are built on top of the deployed smart contracts to manage multi-occupancy buildings. The access delegation is auditable with the usage of blockchain, which is a scalable, trustless, peer-to-peer solution that operates transparently.

- We build a simulator that generates data to synthesize movement of people in a large commercial building. The simulator uses a unified building model created by aligning BOT and Brick ontologies. This model supports path planning and uses an external cost function to prioritize paths. Building usage and access delegation data is generated by the simulator for a specified duration of time.

- We evaluate the proposed access management solution using the simulator.

In summary, this thesis talks about how traditional access management methods can be improved using a decentralized system. We propose the use of blockchain to make access control flexible, decentralized and auditable. To show the usefulness of the proposed solution, we designed the simulator to synthesize real scenarios of a commercial building. The simulator uses the path planning service to find best paths to take in the building to reach a designated room. The simulator helps evaluate the performance of the system in the context of commercial buildings with multiple tenant organizations.

1.5 Outline of the Thesis

The rest of this thesis is organized as follows: Chapter 2 provides an overview of building modelling, current access control methodologies and discusses related work on authorization and access-control solutions developed for the built environment. Chapter 3 describes the methodology in detail for smart-building access control using blockchain smart contracts for managing access. Chapter 4 presents the implementation of the simulator and the services that use graph-based building ontologies for indoor path planning. Chapter 5 evaluates the performance of the proposed solution in terms of throughput and delay. Chapter 6 concludes the thesis by summarizing the contributions, highlighting limitations of the work presented in this thesis, and presenting several directions for future work in this area.

Chapter 2

Background and Related Research

In this Chapter, we discuss existing terminologies and methods for building modelling and access control in the literature, followed by an introduction of Blockchain and smart contracts. After this, we elaborate on various access control methodologies proposed in the literature that use smart contracts and how they differ from our proposed solution.

2.1 Building Modeling

The lack of a common representation for buildings has historically hindered the development of portable building applications. To address this issue, several standards for modelling building data have been conceived in recent years, examples of which are Project Haystack [1] and Brick [7]. The Brick schema [7] defines an ontology for describing the various building spaces and subsystems, their components, and relationships between them. It defines three types of *entities*: locations, equipment, and points. Locations are hierarchically organized, in terms of buildings, floors, and rooms. Equipment may be composed of many parts and may be connected to other equipment with certain functional relationships. They comprise complex building subsystems, such as HVAC, lighting, and plumbing. Sensors and setpoints are two types of physical points that can generate timeseries data and are used in control loops of different equipment.

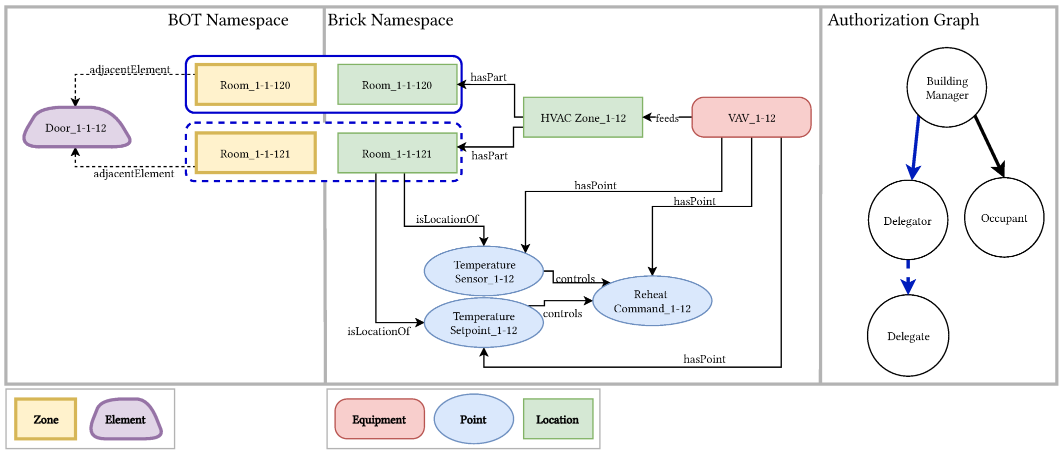


Figure 2.1: The process of aligning Brick and BOT models. A sub-graph of the the Building Graph Model that captures a complete use case in turtle format can be found in Appendix A.1.1.

Brick describes a building through a collection of *triples* (*subject* - *predicate* - *object*), following the Resource Description Framework (RDF) data model. Each triple consists of two entities connected with a relationship, which can be FEEDS, CONTROLS, HASPART, HASPOINT, or ISLOCATIONOF. The collection of such triples forms a directed graph, where nodes represent the entities and edges represent the relationships between them. Figure 2.1 shows a subset of Brick entities and their relationships in an example building. The RDF syntax allows for using SPARQL (the RDF query language) to reason about various entities and relationships. For example, it is possible to retrieve sensors and control points that are located in a specific room or floor of a building, and are used to control the operation of a given Variable Air Volume (VAV) system.

Despite the effort to make Brick expressive enough to capture all important relationships between different building entities, it does not capture the adjacency relationship between different spaces in a building. This information is essential for building applications such as indoor path planning. The Building Information Model (BIM) [57], which supports computational methods for designing and constructing buildings, captures information about the building interior layout and adjacency of rooms, but unlike Brick, it lacks mechanisms for describing functional relationships between different entities [7]. BIM has been traditionally represented using the Industry Foundation Classes (IFC) [8] data model designed to facilitate interoperability in the building industry. The IFC (Industry Foundation Classes) format is a worldwide standard (ISO 16739) for BIM developed by BuildingSmart. In IFC, it is possible to store advanced geometric and semantic information about building components. Entities in IFC are symbolic of physical elements in a building. The 3D geometry of these entities along with their attributes and characteristics are gathered in sets of properties, called P-set. Spatial relations, aggregations and compositions of these entities are also well defined in IFC.

Geospatial Information Systems (GIS) models such as CityGML[33] and IndoorGML[37] embed precise geometric data in 3D and semantic data to identify the internal physical components of a building and their inter-relations. CityGML and IndoorGML are both maintained by OGC (Open Geospatial Consortium). CityGML is a Geospatial standard developed used for rendering, storing, and exchanging virtual 3D models of cities and outdoor facilities. IndoorGML aims at delivering a common framework to represent the interior of buildings and an exchange format for indoor spatial information. The concept of cellular spaces is used in IndoorGML which are indoor spaces identified as a set of cells and provide interpretations about the connectivity between cells. IndoorGML is dedicated to indoor navigation applications and it is topologically far richer than CityGML. However, the use of GIS standards is rather limited compared to the IFC standard, as the geometric format of IFC makes it possible to be converted into other formats.

BIM exhaustively describes composition of building subsystems, it leads to unnecessary complexity when capturing information about a building. This has motivated the Linked Building Data (LBD) Community Group[1] to create the Building Topology Ontology (BOT) [46]. BOT is a minimal ontology for describing the spatial structure of a building. It defines three types of *entities*, namely zones, elements, and interfaces, and captures relationships, such as *adjacency* and *containment* between these entities. Zones are hierarchically organized in terms of sites, buildings, storeys, and spaces, which are spatial

[1] `https://www.w3.org/community/lbd/`

3D divisions used to model rooms. Elements are physical building components such as doors and walls. An interface is the surface where two elements, two zones, or an element and a zone meet. A subset of BOT entities and their relationships are depicted in Figure 2.1 for an example building. Aligning BOT with Brick allows for creating a unified RDF model of building's structure and equipment, thereby enabling the use of SPARQL to reason about the sequence of doors and rooms that need to be traversed to go from one location to another location in a building, and equipment that can be accessed in these rooms.

Several approaches have been proposed that use a BIM model to generate an indoor navigation model to address the challenge of indoor path planning in complex buildings. The path planning method described by Lin et al. [38] extracts both geometric and semantic information about the building components from the IFC data. This information is sampled and mapped into a planar grid which then uses fast marching method(FMM) to find the shortest path in the grid. However, this method requires that geometric and semantic information defined in the IFC be imported into the specific virtual environment manually, which is time consuming and ineffective. Taneja et al. [59] developed algorithms to automatically generate navigation models from IFC data. However, these approaches lack the flexibility needed to be integrated with access management systems as they do not consider sensitivity of spaces on the path.

2.2 Traditional Access Control Methodologies

Access control regulates what resources users may use, based on their assigned roles and privileges. In principle, there are three general mechanisms for reasoning about what permissions should be given to a user. Extensions to these paradigms include risk-aware access control which associates a cost to providing access, and access control using building information models (BIMs) which incorporates the knowledge of building structure and components into access control.

Role-based access control relies on an explicit, and fairly static, list of organization roles associated with privileges. In this model, each user is associated with a role, which entitles them to a set of privileges corresponding to their role(s) [51]. Commercial buildings usually host multiple organizations, each of which defines its own role hierarchy which is relevant to the areas of the building that it occupies; as a result, there is no single role hierarchy that pertains to the building as a whole. Furthermore, often times there is no single central authority who can manage roles for all building spaces.

Risk-based access control is a model where users are granted access to resources based on a scoring function that dynamically and contextually quantifies the risk implicit in this privilege [31]. This approach to access control is more relevant in dynamic environments, where the specific context of the access request should inform whether the request may be honored or not. Buildings are not that dynamic and, in principle, a more explicit, less contextual, access-control mechanism is desirable.

Attribute-based access control grants access rights to users through the use of policies that combine together (with logical operators) different user, resource, object, and environment attributes [60]. In our work, we adopt this paradigm to develop a *cost function* that represents the sensitivity of building spaces based on their function and equipment they contain. A room with many control points, occupied by an employee in a position of authority in the organization, is more sensitive (and is, therefore, associated with a higher cost) than the building's reception for example. In principle, this cost function enables access-control policies to be defined based on sensitivity ranges. It also enables one to reason about the relative sensitivity of spaces and rationalize the access-granting process.

In risk-aware access control, a risk (defined by a cost function) is associated to each user who wishes to access resources. It is then compared against a predetermined threshold, before the user is granted access. This differs from traditional access control models that have predefined policies set for granting access and can be more permissive. The problem arises when "low-risk" users are automatically granted access to resources that were never intended to be accessed by them. In this work, instead of associating risks to users, we adopt a cost function to help develop access control policies where the acceptable risk level can be defined a priori.

Several cost functions have been proposed for access control. Chen et al. [16] propose cost functions for defining risk that incorporate the trustworthiness of a user, the degree of competence of a user with respect to a particular user-role assignment, and the degree of appropriateness of a permission-role assignment for a given role. Salim et al. [50] consider the monetary value of a resource or an inferred impact of misusing it (when monetary value is unavailable) as a basis for their cost function. Bijon et al. [11] propose a risk-based access control framework that incorporates the quantified risk for granting access and specific thresholds calculated based on attributes, purpose, and situational factors. Inspired by these cost functions, we take semantics and relationships of spaces and resources into account when defining the amount of risk associated with accessing spaces and resources therein.

Access control policies for a building can be developed leveraging BIM. Skandhakumar et al. [54] provide a review of spatio-temporal access control models and propose an authorization framework that involves (a) modeling of spatial data in BIM, (b) creation of access policies based on BIM, and (c) authorization of these policies. In particular, the authors introduce 'contains', 'connected', 'adjacent', and 'accessible' relationships between building elements which are accounted for when reasoning about access policies. To capture the relationships between spaces in a building, BIM is transferred to a graph model in [55]. Despite the novelty of this model, it does not incorporate concepts such as sensors, actuators, and building subsystems which can be affected by people who are given access to the building spaces. To specify access control policies, the use of 'eXtensible access control markup language' (XACML) is proposed in [56]. XACML is a standard language for specifying and evaluating access requests. The smart contract based solution proposed in Chapter 3.1 is similar to XACML in that we separate authorization across different services and provide a template smart contract to execute access requests.

Conventional methods for authentication, authorization, and revocation rely on a trusted central authority. For example, the existing authorization method LDAP [62] uses Role-Based Access Control (RBAC) with a single central authority. Kerberos [41] and Jabber [48] are similar in that respect. Several systems are also developed to eliminate the central authority, examples of which are CCN [29] and the Web of Trust [13, 14]. They adopt a decentralized peer-to-peer trust model in which a principal, denoted by a public key, can publish a signature of another public key to denote trust. SmartTokens [20] relies on a token-based access control system for NFC-enabled smartphones, in which the delegation of access to other smartphone users can be accomplished without a central authority. Although SmartTokens uses symmetric cryptography, users need to present all delegated tokens through the delegation chain in order to be verified. In recent work [36], a lightweight distributed authorization protocol is proposed supporting delegation of access right to a smart device in the form of a Bloom filter. This method of delegation uses secured hashing to prevent the permission from being forged.

2.3 Blockchain and Smart Contracts

Blockchain is a distributed and shared ledger that serves as an irreversible and incorruptible public repository [40]. It enables the occurrence of a particular transaction without requiring a central authority. Compared to traditional database systems, it offers three major advantages:

- As a *distributed* system, blockchain eliminates the need for a costly infrastructure that relies on prox cards for occupants and requires one of these occupants (or dedicated security personnel) to escort visitors to their meeting locations using their own prox cards.

- Blockchain does not require a trusted third party to certify transactions, thanks to public-key cryptography and a consensus mechanism. This allows digital transactions to occur between parties that do not have pre-established trust relations, i.e., *trustlessness*.

- The state stored in blockchain is *immutable* due to the use of cryptographic hash functions.

Blockchain is analogous to a log whose records are group of transactions that have a timestamp. Each block has a cryptographic hash and it references the hash of the block that was created before it. Any node with access to this series of blocks can read the global state of the data that was exchanged on the blockchain network. Blockchain is called a trustless network because the parties involved can transact even without trusting each other, which means faster reconciliation between the parties. This is due to a key characteristic of blockchain networks, the use of cryptography. Users interact with the blockchain using their private and public keys- being addressed using their public key and using their private key to sign their own transactions. The neighboring nodes validate and broadcast the transaction, spreading the transaction to the entire network. The validated transactions are collected into a timestamped candidate block, this process is called mining.

If any node can join the blockchain, then the network is called a public or permission-less network, for instance Bitcoin, whereas if we have a whitelist of nodes that can join, then it is a private or permissioned network, like Ethereum. The type of blockchain being used decides the consensus mechanism of the network. Because of the Sybil attack [22], consensus in public networks is costly and an monetary incentive, or cryptocurrency, is given to the miners. Private networks are used in a supervised, regulated environment and provide a higher throughput than a public network could offer. Since the participants are restricted, costly consensus mechanisms are not needed as the risk of a Sybil attack is ruled out, which means there is no requirement for a monetary incentive for mining. Private blockchain mimics a distributed database under a decentralized administration which provides improved transparency and auditability across the involved nodes than in traditional distributed databases [53].

In order to avoid a Sybil attack and facilitate trust between nodes in a public blockchain, consensus mechanism are made competitive and computationally expensive. Mining is made computationally expensive by making miner node find the random number (nonce) in the block's header, which generates the proof-of-work (PoW) [39]. An alternative to PoW is called proof-of-stake, which is not as expensive. For a miner node to choose the next block, it should hold a value of stake (balance) which is proportional to the stake of the network value that each node holds.

The absence of a central authority in blockchain could lead to conflicts in the global state of transactions. For this, the nodes need to reach consensus on which transaction is valid and conforms to the rules of the network. These rules are defined for each blockchain network and are stored into each blockchain client. The validity of the incoming transactions and whether it should be communicated to the network or not is decided using these rules. There are various distributed consensus protocols, a popular one being based on state-machine replication with byzantine fault tolerance [15], which can function successfully even in the presence of certain number of malicious or faulty nodes.

The smart contract concept was originally proposed by Szabo in 1994, who states that "The general objectives of smart contract design are to satisfy common contractual conditions, minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries [58]". Parties can use smart contracts deployed on a blockchain network to perform trustless transactions without any intermediaries as the blockchain provides a great infrastructure because its transactions are transparent and traceable. Smart contracts are simple computer code written in a language supported by the underlying blockchain platform. This code is automatically executed in accordance with the designated triggering conditions in the contracts, the conditions being outcomes of transactions, external events or interactions with other smart contracts.

The smart contract executes code on a blockchain to facilitate, execute, and enforce the terms of an agreement between different parties, and are entirely managed by the code, not subject to control of any central entity. It can also be used to encode an arbitrary state-transition function. It executes independently and automatically in a manner defined inside the contract on every node in the network, according to the data that was included with the transaction triggering it. Once deployed on the blockchain, the contract is immutable. The only way to mutate a deployed smart contract is to create and deploy a new one with the consent of all transacting parties. The smart contract is a deterministic piece of code, where the same input will always return the same output and it has its own state and account in the blockchain. Each contract is assigned a unique address. Users can send a transaction to this address for execution. A callback function is executed when a transaction execution request is received. If the transaction is successfully completed, the contract's state is updated. Otherwise, any changes made to the state are reverted.

The development of smart contracts is supported by many blockchain platforms, Ethereum [12] being the most well known and, perhaps, the most broadly adopted one. Ethereum provides an abstract foundation layer for smart-contract development: a blockchain with a built-in Turing-complete programming language for the specification of smart contracts, with arbitrary rules for ownership, transaction formats and state-transition functions. Ethereum has its own cryptocurrency called *ether* and an internal currency to pay for computations and transaction fees called *gas*.

2.4 Access Control via Smart Contracts

Drawing on [3] that argues for democratized access to the physical resources in buildings using a blockchain, we utilize a private Ethereum blockchain to store the *authorization graph* of a specific building. Executing a transaction (e.g., for adding or revoking users' accesses) leads to a state change and updates this graph. The delegation of trust can be performed by any user without communicating with a central authority. The state can be read from the blockchain to verify access for any user at any time.

Our implementation uses a private Ethereum blockchain over a public blockchain for performance and privacy reasons[2]. The private blockchain net-

[2]Private blockchains can achieve better scalability than public blockchains, thanks to

work is comprised of several nodes representing different groups within an organization or different organizations housed in the same commercial building. Compared to a centralized access control system which runs a private server, it provides better transparency, availability, and robustness; there is no single trusted entity and no single point of failure, the integrity of access-related data is always maintained and tenant organizations can easily audit transactions. Although privacy is easier to achieve in centralized systems, the private blockchain helps preserve privacy to some extent as only specific nodes within the organization are allowed to connect to the network. We note that private permissioned blockchains, such as the Hyperledger Fabric, and hybrid blockchains could provide advantages similar to a private Ethereum blockchain and can be considered as alternative solutions.

To tackle the privacy issue, Kosba et al. [34] build a tool, called 'Hawk', which helps developers create privacy-preserving smart contracts without the need of cryptography. The tool is responsible for compiling smart-contract code to a privacy-preserving version. Watanabe et al. [61] propose encrypting smart contracts before deploying them to the blockchain network so only those participants who have the key can access the contract's content (i.e., the state). Bernable et al. [10] provide a comprehensive review of privacy preserving blockchain approaches.

RBAC-SC [17] performs role-based access control using blockchain and a challenge-response protocol for authentication. FairAccess is a cryptocurrency blockchain-based access control framework [43]. This work is different from ours as a distinct smart contract is created for the access control policy of every resource-requester pair. Furthermore, they include the IoT devices in the blockchain, whereas there are IoT devices that do not have the capabilities to run the blockchain on them. In another line of work, a distributed architecture called ControlChain is proposed [45]. ControlChain enables the expression of a wide variety of access control models, such as RBAC [23], OrBAC [30]

the limited number of participants they have, and can minimize privacy concerns since only authorized users are allowed to connect and perform transactions. Furthermore, the transaction cost can be less of a concern as it is not tied to the volatile cryptocurrency market.

and ABAC [28], deployed on IoT. This mapping is enabled due to the use of a *Decoder* entity that automates the translation of access control model and rules to their supported mechanisms. However, none of the these decentralized systems addresses how building metadata can be linked with smart contracts to extend access control to spaces and equipment in the building.

Wave uses the concept of entities, namespaces and delegation-of-trust (DOT) to manage access of resources [3]. An entity is simply a key pair, identified by its (public) verifying key, that may represent any participant: individuals, devices, services, applications, components of the system implementation, and so on. A namespace is a hierarchy of resources that is identified as and owned by its authorizing entity, which has full access to all resources within the namespace. Each resource is identified by a path, like namespace/path, which is rooted in the namespace identity. The kind of permissions an entity can have include the ability to publish to a resource or subscribe to a collection of resources in a namespace. In general, a sensor device publishes to the resource that represents it; an actuator subscribes to a resource expression representing its interface. Delegating access to an entity can be done when it is offline, which is not the case with conventional methods. When an entity publishes (or subscribes) to a resource, it must present a proof of authorization consisting of a valid DoT path in the permission graph from the authorizing entity of the namespace to itself, encompassing the resource. The same transparency is achieved by our work. WAVE also supports out-of-order and non-interactive delegation, which is replicated in our smart contracts.

In [5], the authors' experiments reveal that a Blockchain-based access control system will not scale to a global size. They state that Blockchain introduces about a minute of latency when adding objects to Ethereum. However, with a private blockchain for a small number of buildings, our solution works well as described in our implementation since managing permissions is infrequent as compared to accessing permission data. Also, the calendar application helps provide access well in advance, reducing the effect of latency. To address the scalability issue that may arise in a large campus comprised of several commercial buildings, other access control methods could be considered in future work.

BOSSWAVE [4] builds on WAVE and uses Ethereum as its foundation for a public ledger to provide democratized access to the physical resources in buildings. The read/write permissions are protected and can be accessed by entities that received the delegation of trust. Our system is similar to WAVE in that they both leverage blockchain technology. However, using smart contracts for authorization to spaces and doorways apart from building subsystems and equipment is novel in this work. We develop APIs to help interact with the smart contracts; this will define fine-grained access control for any type of user.

In the next Chapter, we discuss how smart contracts deployed on a private blockchain network are used for flexible access management. We elaborate on the three smart contracts designed to facilitate different functions for access control.

Chapter 3

Flexible Access Management Using Blockchain

In this chapter, we describe how access privileges are stored in a blockchain using smart contracts and how access is verified when a visitor tries to access a physical space through a locked door.

Consider a simple application scenario, where a group of individuals are invited to a meeting that takes place in a specific room of an office building. The process starts with a calendar invitation created by the meeting host, listing the invitees using their email addresses, the room where the meeting is to be held, and the time when it is to take place. The first step is to examine whether the invited individuals are known to the access-control service: if not, new entities need to be created to represent their credentials (Section 3.1.1).

Next, the possible paths in the building to reach the meeting room are computed. We only consider the problem of indoor path planning, from a main building entrance to a meeting room; nevertheless, our methodology can be extended to the overall problem of planning a path from each individual's location, taking into account their route preferences and parking needs, as long as the relevant information is captured by some geospatial model. Each of these indoor paths is presented to the meeting host along with a cost that represents the overall risk of giving the invited individuals access to the rooms located on this path and the building elements (equipment, sensors, and setpoints) that contribute to it. The host reviews the possible paths and selects the preferred pathway, which is by default the path that has the lowest cost. The selected pathway is then transformed into a smart contract (Section 3.1.2), where the meeting host delegates each invitee with the necessary privileges to access the various doorways on the path, and a personalized QR code (akin to a boarding pass) is generated to represent each contract. These QR codes are shared with each participant in the calendar invitation.

Before the meeting occurs, each meeting participant arrives to the building and scans their individual QR code to open doors on the path from the building entrance to the designated meeting room. Each QR code scan invokes a request to the Ethereum ledger (Section 3.2) to verify that the bearer is authorized to access the corresponding door at the present time, given that they have already opened some other doors in a the order pre-specified by the indoor path selected by the meeting host. If the meeting host has access to these rooms and had authorized the meeting participants to access these rooms before the meeting, the access is verified and a control signal is sent to the actuator, i.e., the electric door strike, to open the door. The process repeats for each door along the path. We elaborate on these steps in the following.

Our access-control methodology involves granting, revoking, and verifying user permissions to access rooms and equipment therein using smart contracts. To achieve a thorough reasoning about sensitivity and security of the spaces to which access needs to be delegated, we use the following three steps: (1) creating a unified RDF graph of a building by aligning the building's BOT and Brick models; (2) identifying all possible paths between two locations using a graph traversal algorithm which is implemented by a sequence of SPARQL queries; (3) determining the cost of each path by running a number of SPARQL queries; The unified RDF graph of the building is created by identifying common syntactic entities in the Brick and BOT model. These entities that represent the same type are joined to form a new class of entities such that it relates to elements in both the graphs. This is described in detail in Chapter 4. The access management methodology stated above are implemented using three services [18]:

- **Building-Representation Service** determines the cost of an indoor-path using information captured in the Brick model, described in Section 4.2.3.

- **Path-Planning Service** relies on the BOT model to find all possible paths between two given locations. These paths are presented to the user along with their costs, enabling the user to choose a desired path. Path-Planning Service is described in Section 4.2.

- **Access-Control Service** specifies a smart contract, given a delegator, a delegate, a path corresponding to a sequence of building spaces and doors between them, and a time period during which the delegate should be able to access the resources on this path. It also handles validation of a delegate's authority to access a door, at run-time.

3.1 Smart Contracts

In this section, we describe the prototype implementation of our access control service that grants, revokes, and verifies user permissions through smart contracts deployed on our own private blockchain using the Ethereum network. Ethereum is a blockchain platform that includes a Turing complete scripting language called *Solidity* [2] for building, deploying, and implementing smart contracts. These contracts have no restrictions in terms of size and are stored in the blockchain. Our methodology relies on three smart contracts, namely *Archives*, *Implications* and *Exclusions.* They are defined as follows:

- The **Archives** contract manages and stores the entities and access rules for each user. Any creation or deletion of entities or access rules leads to a state change. The state of the Archives contract is read to verify access.

- The **Implications** contract cross-verifies the validity of access rules and processes the path resources that should be accessible to an entity for a specific access rule. A state change takes place only when a new rule is created.

- The **Exclusions** contract is used when the delegator chooses to provide a list of all the resources that the delegate is not allowed to access, even though they are in spaces that the delegate is allowed to reach.

These smart contracts are compiled and deployed using scripts. For each contract, a unique bytecode and contract address is created. The contract address is used to make a transaction to a contract in the private blockchain network, either from external functions or from another smart contract. An API, illustrated in Figure 3.1, enables the client applications, such as our meeting-planning example scenario, to interact with the deployed smart contracts. The API exposes a number of operations, receiving (or producing) JSON objects as input (or output).

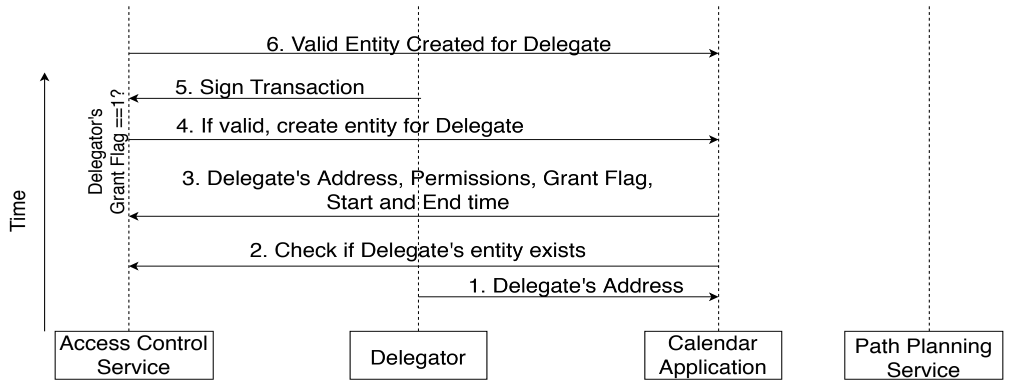
3.1.1 Adding a New User

As shown in Figure 3.1a, the delegator, the meeting host, provides the calendar invitation of the meeting to the Calendar application. The delegate here is the meeting participant, who could be an external visitor or an existing occupant of the building. The delegate's email address can be used as the unique identifier for the delegate's entity. This address is used to verify if an entity exists already for the delegate, or a new entity needs to be created by the Access-Control Service[1]. The start and expiry times for the entity could be taken from the calendar invitation or also be provided by the delegator. If no expiry time is specified, the implication is that the contract is valid for the foreseeable future until it is explicitly revoked. In our application scenario, meeting participants should only have access to the building for a specific time period. Hence, the expiry time must be provided.

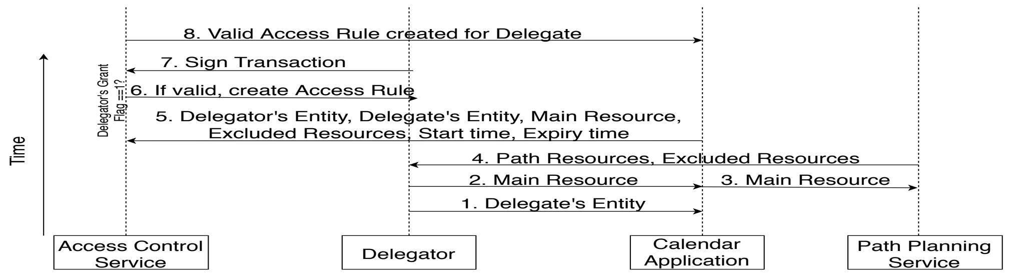
In addition to the delegate's address, and start and expiry times, the set

[1]Note that there are commercial solutions for decentralized identity management such as Civic (`https://www.civic.com/`), Sovrin (`https://sovrin.org/`), and uPort (`https://www.uport.me/`). These solutions allow for users to manage their own identities, and can be integrated with our system to eliminate the need for independently managing and verifying the identity of building occupants and visitors. However, due to the commercial nature of these solutions, we do not use them in our implementation. As a future direction, it would be interesting to see the performance impacts of using a third-party decentralized identity management service.

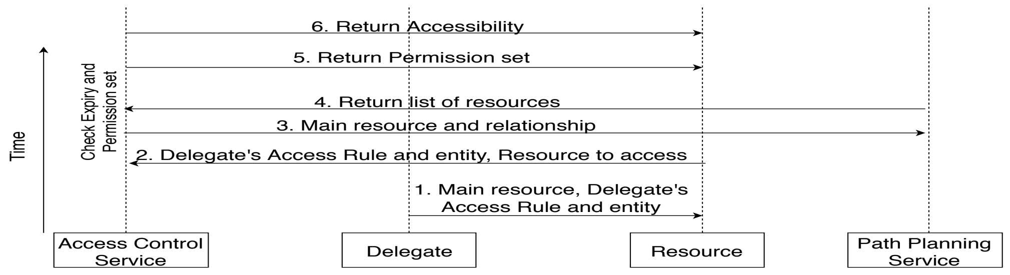
of permissions that the delegator intends the delegate to possess must be provided as input to the API. These include read (for accessing sensor measurements) and write (for changing the value of a setpoint) permissions for points and equipment resources. The permissions set also includes a flag to indicate whether the new delegate entity is allowed to further delegate access to other users. The meeting-planning application interacts with the Access-Control Service, providing the required information as depicted in Step 3 of Figure 3.1a. This information is sent to the Archives contract to create a new entity. The contract checks validity of the input and creates an entity for the given delegate's address.


6. Valid Entity Created for Delegate
5. Sign Transaction
4. If valid, create entity for Delegate
3. Delegate's Address, Permissions, Grant Flag, Start and End time
2. Check if Delegate's entity exists
1. Delegate's Address
Delegator's Grant Flag ==1?
Time
Access Control Service
Delegator
Calendar Application
Path Planning Service

(a) Adding a new Entity.


8. Valid Access Rule created for Delegate
7. Sign Transaction
6. If valid, create Access Rule
5. Delegator's Entity, Delegate's Entity, Main Resource, Excluded Resources, Start time, Expiry time
4. Path Resources, Excluded Resources
2. Main Resource
3. Main Resource
1. Delegate's Entity
Delegator's Grant Flag ==1?
Time
Access Control Service
Delegator
Calendar Application
Path Planning Service

(b) Adding a new Access Rule.


6. Return Accessibility
5. Return Permission set
4. Return list of resources
3. Main resource and relationship
2. Delegate's Access Rule and entity, Resource to access
1. Main resource, Delegate's Access Rule and entity
Check Expiry and Permission set
Time
Access Control Service
Delegate
Resource
Path Planning Service

(c) Verifying an Access Rule for accessibility information.

Figure 3.1: Three APIs for smart-contracts

Once all the checks of the smart contract are successful, the delegator has to sign the transaction using their private key. Any failure in the transaction would revert all the changes made. If a valid response is not returned from the smart contract, the entity is not created for the delegate. The API reads the transaction hash received from the contract's callback function and finds out whether a valid entity was created for the delegate. Every entity created has a unique address assigned to it.

3.1.2 Adding a New Access Rule

After creating a new entity for the delegate, the delegator can proceed to create an access rule, based on the spaces and equipment resources that the delegate is authorized to access. In our application scenario, the meeting participants must be allowed to access all spaces they need to go through to reach the meeting room, and all sensing and control points located in this meeting room. This is exactly the information that the path-planning queries Q1 and Q3 deliver (refer to the discussion in Section 4.2.3). If the delegate is already known to the system and has previously had access to some resources in the building, access must only be provided for the additional spaces that are on the selected path.

The delegator has two options with respect to the various points located in the destination room: (a) they may authorize the delegate to access all of them, except an explicitly excluded set, or (b) they may include all of them in the contract (default behaviour). The Exclusions contract is used to implement the former. The permissions set mentioned in the previous section determines if the delegate can read sensor measurements or write control setpoints in the destination space.

The smart contract's function requires information such as the delegator's and delegate's addresses. The contract verifies the entities through these addresses. In addition, the contract checks if the delegator's entity has the permission to grant access to new users.

For each access rule, the Path-Planning Service provides a list of resources for the delegate to access as shown in Figure 3.1b. This list is generated when the delegator selects a path (likely the one that has the lowest sensitivity cost). The associated list of path resources and inaccessible equipment (that could be specified by the delegator) is stored with the Implications and Exclusions contracts. These contracts identify a unique access rule using its hash. The two contracts are called from the Archives contract when a new access rule is being created to add the path resources and excluded equipment list. In our application scenario, the Implications contract defines the order in which the visitor should access the doors to reach the meeting room. But if access is being granted for an occupant, there needs to be no restriction on the order of the path that should be followed. This processing is also done by the contract apart from storing the list.

Other parameters required by the contract's function include the start and expiry times for the access rule being created and the main or destination resource, as in Step 5 of Figure 3.1b. All the fields are packaged into the request in JSON format. The API runs a callback function to the smart contract to create the new access rule with the provided fields. The access rule is uniquely identified with a hash created using the delegator and delegate's information. The Archives contract is called to create the access rule. It verifies the entities, checks the granting rights of the delegator, and also validates that such an access rule was not defined earlier and is a new access rule for the delegate.

A valid access rule is created with the required fields as shown in Steps 6-7 the delegator signs the transaction using their private key. If the response returned from the smart contract is not a "success" value then the new access rule for the delegate is not created. The API reads the transaction hash received from the callback function of the contract and responds if a valid access rule was created for the delegate.

3.1.3 Verifying User's Access Privileges

While accessing a resource, the existing entities and access rules are read from the smart contract. It is important to note that the run-time verification of the user's credentials simply queries the private blockchain and does not cause a state change. Thus, it does not require any "gas" (i.e., has zero transaction fee). However, creating a new entity for the delegate or adding new access rules costs a specific amount of “gas” since they result in state changes on the private blockchain.

When the meeting participant arrives to the building and tries to access a specific resource, e.g., the meeting room, a request is sent to the API endpoint to verify if they indeed are authorized to access this resource. As in Steps 1-2 of Figure 3.1c, the API request requires the entity and access rule of the delegate, the resource name, and the action to perform. The state of the smart contract is read to retrieve the details of the entity and access rule for the delegate. The functions of the smart contract check the validity of the entity and access rule, and verify its start and expiry times. Next, the permission set is inferred from the entity, the main resource, the implication and excluded resources are found from the access rule. This process is described in the Figure 3.1c.

After receiving this data from the contract, the requested resource is looked up in the implications and excluded resources. If it is not found, a query is invoked (Q3 in Section 4.2.3) to identify all the resources related to the main resource with the required type of relationship, shown in Steps 3-4 of Figure 3.1c. If the requested resource is included in the returned resources, then the delegate has access to the requested resource. Otherwise, the delegate is not authorized to access the requested resource.

If the resource is equipment or a point in Brick, then the permissions from the delegate’s entity are verified to see if they can read or write (control) the requested resource. Once the validation process is completed, the API returns a JSON response indicating whether the requested action is allowed or not. Since these actions do not need to be mined, they are performed immediately without any delay.

Similarly, whenever a visitor enters the building through the main door, they have to follow the path to which the access was provided. Hence, the delegate has to follow a sequence of doors in order to reach the destination room. For this, the smart contract function reads the set of implications that lead to the destination room. The implications are stored as in the order of the path. This leads to verifying a sequence of access requests.

A user pointer is used to store the current location of the delegate. When a delegate tries to access a resource present in the set of implications, the pointer's current value and the requested resource are verified in the sequence of the path. If the order of requested access matches the path, the delegate is allowed to unlock the door through hardware control of the smart lock. If there is a mismatch, it would imply that the delegate has either tried to enter a wrong door or has skipped a door on the way. This method enforces that the delegate follows the path he is supposed to and not get lost on his way to the destination.

The time to access the implications should also be considered when specifying the start time. For instance, access could be provided to the delegate 30 minutes in advance so that they can come in early and reach the designated room. Requests invoked to check access for a user at different stages are logged. It is also possible to add new smart contracts for tracking the number of times the state of the contract is read to verify access to a certain zone, and the users who requested access.

3.2 Deployment and Real-time Access Control

Smart contracts are programmed using Solidity, a specialized Turing-complete language. The following code snippet shows the smart contract responsible for adding a new entity resource in the system:

```
...
function AddEntity(bytes32 sid, uint expiry,
bytes32 hash, uint permission,bool grant) public payable
{
  if (Entities[sid].validity == Validity.Void &&
                expiry > block.timestamp) {
    Entities[sid].grantFlag=grant;
    if(permission == 0)
        Entities[sid].permission.push(Perms.Read);
    if(permission == 1)
        Entities[sid].permission.push(Perms.Edit);
    Entities[sid].validity = Validity.Valid;
    Entities[sid].expiry = expiry;
```

```
        Entities[sid].hash= hash;
    }
    return;
}
...
```

The function follows the contract rules before adding a new entity into the system. It checks whether the entity is known to the system before hand and does not add a new entity if it already exists. It verifies the permissions and access duration for the user being added to the system and then creates a new unique entity if all conditions are satisfied. Each entity has a unique identity. Similar functions exist to add unique access rules and to verify the entities and access rules. Withdrawing access or making an entity void can also be done by entities that have the permission to grant access to new users.

The infrastructure that the access control system uses is implemented on servers running Ubuntu 18.04. Ethereum blockchain node is deployed on the server by downloading and installing Ethereuem's geth 1.5.8. All the smart contracts are deployed on one node of the blockchain network. Some or all of the nodes could be used as mining nodes of the blockchain network. The interactions between the calendar application and the deployed smart contracts are accomplished with HTTP requests made to the address of the deployed contracts on the blockchain network. All of the queries are implemented via web3.py and it communicates to the smart contract's address through Flask APIs. The Access-Control Service is packaged as a docker container which can be downloaded from github[2] and run on any client machine. This helps the client to make calls to the smart contract's state to verify entities and access rules for meeting participants.

Upon selecting an indoor path that meeting participants can follow to reach the meeting room, the smart contract is initiated by the Access-Control Service to authorize the participants to open the required sequence of doors and access certain spaces shortly before the meeting starts. Users who already have access to (some of) the building's systems and spaces can delegate access to other individuals. This creates a tree-like structure of an authorization graph, as

[2] `https://github.com/leepakshi9/SmartContractsAccessManagement`

depicted in Figure 2.1. This implies that the process has to be bootstrapped with some original space manager(s), e.g., the Building Manager in Figure 2.1. Traditionally, they might be the building-security personnel, or (some of) the long-term building occupants.

In our application scenario, the meeting host (the delegator in Figure 2.1) is assumed to have access to the whole floor (including *Room_1-1-120* and *Room_1-1-121*) and can thus delegate access to the spaces and doorways in the path leading to the meeting room (i.e., *Room_1-1-120*). Hence, the delegator grants the meeting participant (the delegate in Figure 2.1) access to all the doors and rooms in the path they have selected, which implies that all the rules mentioned in the smart contracts should be satisfied for the delegation to succeed. To that end, the delegator first uses the smart contract to create a new unique entity for the delegate, assuming that she is not already known to the system (otherwise, the existing entity is used to add a new access rule). Each entity is a node in the authorization graph, therefore creating a new node for the delegate. The delegator provides the delegate's address, which is the public key of the delegate, along with the sequence of permissions implicit in the chosen path and the period during which these permissions should be valid, in effect, a period covering the meeting duration. If no failure is encountered in the creation of entity and all the requirements of the contract are fulfilled, a valid entity is created for the delegate.

Next, a new access rule has to be created, containing the delegator's entity (the meeting host) as the source, the delegate's entity as the target, the expiry time after which the access rule will become invalid, and the list of resources that the delegate is allowed to access. This creates the relationship between two nodes, the delegator and the delegate in Figure 2.1, i.e., the delegator and the delegate, which describes the access rule. The list of resources received from the Path-Planning Service are stored in the order they should be invoked, starting from the building's main entrance and concluding with the meeting room. At run time, the in-order access of the list of locations and doors is evaluated, which helps the delegate to navigate in the right path. The destination and equipment that can be accessed by the delegate are also stored.

Once all the requirements to create a new access rule in the smart contract are fulfilled, a successful transaction is executed. A failure would revert any changes that were made to the contract's data.

An important advantage of smart contracts is the flexibility they afford in the revocation process. Once a smart contract has been issued to authorize a delegate with access to some spaces, it cannot be deleted; the blockchain is immutable and anything stored on the blockchain as a transaction cannot be deleted. To revoke this authorization, the access rule assigned to that user must be made invalid. Similarly, to remove a user from accessing any system or space in the building, for instance when an employee leaves the organization, the entity belonging to that user is made invalid. This is enough to make sure that this user can no longer access building spaces and equipment therein. Once a rule or entity is made invalid, it can no longer be reused for the same address of the delegate.

3.2.1 Checking Credentials at Run-Time

Upon arrival, when the delegate tries to access any space or equipment, the data from smart contracts need to be read by the Access-Control Service to validate whether the delegate's credentials authorize them to access the spaces they attempt to access. To that end, the access rule assigned to the delegate is read from the blockchain. From this data, the duration and validity are first checked. If the access rule is valid, the set of accessible resources assigned for the delegate are checked and the permissions granted to the delegate, as a part of the access rule, are verified. If the requested action is included in the set of permissions and the requested resource is included in the set of accessible resources, then the user will be allowed to access the requested resource. Similar is the case for accessing points and equipment. To read or modify the temperature setting in the room, the accessibility needs to be checked and the smart contract decides if the requested action can be taken.

3.3 Summary

In this chapter, we discussed how blockchain and smart contracts can be utilized as a decentralized system for access management. We also described in detail how Access-Control Service runs on top of blockchain to help delegators and delegates transact with the smart contract and blockchain. In the next chapter, we talk about a use case of an office building to describe how access can be managed in a commercial building by simulating presence, movement and occupancy of people in the example building.

Chapter 4

Case Study: Access Control in an Office Building

In this chapter, we use an example building to describe how access can be managed in a real multi-tenant commercial building. The floor-plan of the example building is illustrated in Figure 4.1. A more comprehensive solution is needed so as to achieve the scenario of managing real buildings. This requires, in addition to managing access rights, reasoning about what spaces should be accessible to whom. In the following sections, we describe in detail how we create a unified RDF graph of the example building by aligning the building's BOT and Brick models, identify all possible paths between two locations in the building and determine the cost of each path.

4.1 Functional and Physical Modelling of an Example Building

This section describes our methodology to create the BOT and Brick models of the building (if they do not exist already) and align them to create a RDF graph queryable via SPARQL. This RDF graph represents the relationships defined in both Brick and BOT. It allows for identifying pathways in the building using a graph traversal algorithm and quantifying the cost-sensitivity of each pathway.

Figure 4.1 depicts the floor plan of an example commercial building occupied by a single organization. We manually converted this floor plan into a

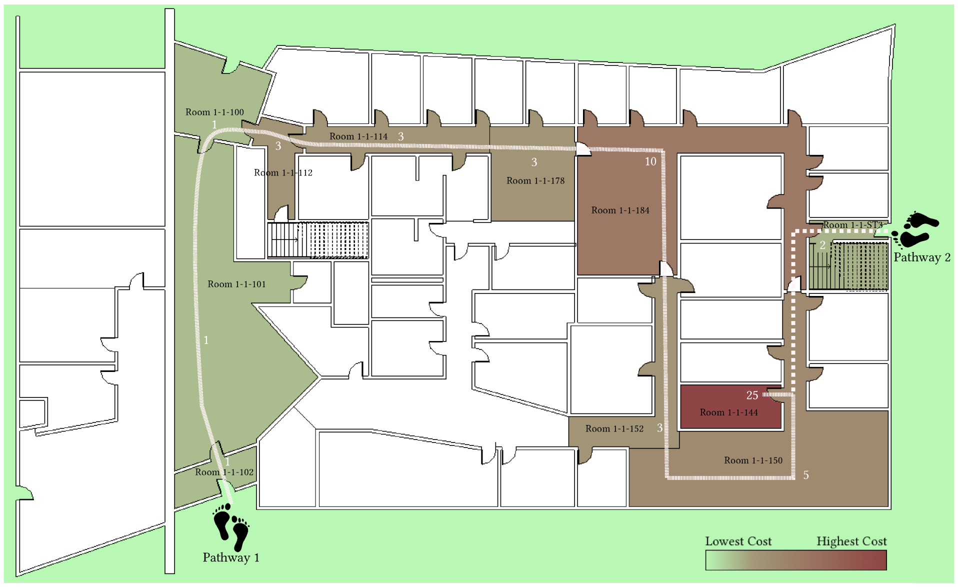


Figure 4.1: A Building's floor plan showing 2 example pathways to a meeting room and the cost-sensitivity of every room on these pathways.

RDF graph model of the building based on the BOT ontology[1]. This model describes the building's topology in terms of where rooms, doors, walls, and other physical elements are located and their adjacency relations. The left panel of Figure 2.1 shows a small subset of nodes in the resulting RDF graph: two rooms of type ZONE (subclass of SPACE) and a door between them of type ELEMENT. Each room is connected to the door with the ADJACENTELEMENT relationship.

Similarly, we created the Brick model of the building in RDF. This involves extracting point names and their types from the BMS, inferring functional relationships between different equipment, and identifying their location from blueprints of the building. Both the RDF graphs of Brick and BOT ontologies are stored in HodDB [24], which is a database and a fast query processor for Brick models. HodDB contains a query processor, which accepts SPARQL queries and processes them by accessing Brick and BOT ontology data that is stored in a LevelDB.

The middle panel of Figure 2.1 shows a small subset of nodes in the re-

[1]Generating this model can be automated by using information from building information modeling software.

sulting RDF graph: two rooms and a HVAC zone of type LOCATION, a VAV system of type EQUIPMENT, and a temperature sensor, a temperature setpoint, and a reheat command of type POINT. The VAV system is connected to the HVAC zone with the FEEDS relationship, as it supplies air to this zone. The HVAC zone is comprised of the two rooms, so it is connected to them with HASPART relationship. The temperature sensor and setpoint are located in one of these rooms and are therefore connected to it via ISLOCATIONOF relationship. The reheat command is computed based on the difference between the measured and setpoint temperatures and is used to actuate the VAV system; thus, the temperature sensor and setpoint are connected to the command via CONTROLS relationship, and the VAV is connected to all three of them via HASPOINT relationship.

Once the Brick and BOT models are created, the next step is to align the two models that is to ensure the entities corresponding to the same building location (e.g., room) in Brick and BOT models are the same in both sets of triples. Otherwise, it would be impossible to reason about which pathway enables access to which equipment, which is necessary for establishing our cost function as discussed in Section 4.2. To this end, we identify syntactic entities that represent the same semantic entity in the two graphs and join them to create a new entity. This new entity is a subclass of LOCATION in Brick and SPACE in BOT and therefore can be connected to entities defined in both Brick and BOT. The borders around *Room_1-1-120* and *Room_1-1-121* in Figure 2.1 show the new entities created by merging respective entities in Brick and BOT models.

4.2 Path Planning using Brick and BOT

In this section, we discuss how the Path-Planning Service identifies all possible paths between two locations using a graph traversal algorithm which is implemented by a sequence of SPARQL queries. Further, we talk about determining the cost of each path by running a number of SPARQL queries. The Path-Planning Service is packaged as a docker container and is available on github link provided in Section 3.2.

The issue of facility security is quite complex, especially when buildings house expensive or sensitive equipment. Intuitively, the characterization of the sensitivity of a space in terms of standard zones defined in Section 4.2.1 considers the activities taking place in the space and possibly the role of its occupants, but it does not consider the equipment housed in, or accessible through, the space. This is why, our access-control methodology proposes a composite sensitivity function for each space that integrates information about (a) equipment and subsystems a user may be able to potentially control by accessing this space, (b) sensor readings a user may be able to read by accessing this space, and (c) the security zone classification of this space according to Table 4.1. Through this sensitivity quantification, we aim to help a delegator, such as the meeting host in our example scenario, make informed decisions about providing access to the building spaces.

The RDF model of the building can be used to annotate each building *location* with its *sensitivity level*. We assume that there are some broadly shared and agreed-upon principles for quantifying the sensitivity of locations. This is a realistic assumption in this domain; in our work, we have adopted the 'Hierarchy of Zones' as described in the 'Operational Security Standard on Physical Security' of the Government of Canada[2]. We note that similar specifications exist in several other countries, such as the United States[3] and New Zealand[4].

4.2.1 Defining Sensitivity of Security Zones

The Canadian standard defines five zones, as seen in Table 4.1. Access to *public* zones, such as the grounds surrounding the building, do not need to be controlled. *Reception* areas may be inaccessible to visitors, except during specific times of the day or for specific reasons. Access to *operations* zones

[2]https://www.tpsgc-pwgsc.gc.ca/esc-src/msi-ism/chap4-eng.html

[3]https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodm/522022M.pdf

[4]https://www.protectivesecurity.govt.nz/physical-security/understand-the-physical-security-lifecycle/design/apply-good-practices/security/

Table 4.1: Zones and their corresponding requirements

Requirements	**Security Zones**				
	Public (0)	Reception (1)	Operations (2)	Security (3)	High Security (4)
Monitoring		×	×	×	×
Screening Required			×	×	×
Clearly Separated				×	×

is limited to personnel who work there and to properly escorted visitors. *Security* areas are limited to authorized personnel and to authorized and properly escorted visitors. Finally, *high-security* areas are limited to authorized, appropriately-screened personnel and authorized and properly-escorted visitors. The standard advises that both *security* and *high-security* zones should be monitored 24 hours a day, and that zone levels should be accessed in order, i.e., a high-security zone can only be accessed from a security zone.

We use the Zone entity in BOT to model different security zones that exist in a building. Consider for example the rooms 1-1-100, 1-1-101, 1-1-102, 1-1-112, 1-1-114, 1-1-144, 1-1-150 in Appendix A.1.3. Rooms 1-1-102, 1-1-101, 1-1-100 are considered as reception zones, since they are the locations that visitors can access when coming from a public zone without any credentials. Room 1-1-112 is considered an operations zone, as only employees are allowed to access the area with proper credentials. Connected to room 1-1-112 is room 1-1-114, which is considered to be a security zone, as it is physically restricted from room 1-1-112 and additional credentials are required to access the area. It should be noted that room 1-1-114 must be accessed from room 1-1-112, as a security zone should only be accessed through an operations zone. Room 1-1-144 is an example of a high-security zone that must be accessed from room 1-1-150 which is classified as a security zone.

To quantify the sensitivity of the five security zones mentioned above, we map them to an ordinal scale of 0 to 4, with 0 being a public zone, which is not at all sensitive, and 4 being a high-security zone, where access should be carefully controlled. We choose the ordinal scale because the order of security zones signifies their relative importance (e.g., a high-security zone, labelled 4, is more important than a public zone, labelled 3). Thus, each room/space in the building is associated with the cost of its zone.

```
SELECT ?entity ?x
WHERE {
    ?entity rdfs:subClassOf ?x .
    ?x rdfs:subClassOf brick:Point .
}
```

Figure 4.2: We obtain all the point types based on the following SPARQL query which obtains the first sub-class of the brick point class in the graph model

4.2.2 Defining Sensitivity of Equipment and Points

Building subsystems and points should have different sensitivity costs assigned to them. This is because some subsystems may control more critical aspects of a building (e.g., the lighting system is less critical than the HVAC system), and some points in a subsystem may be more important than others (e.g., using the thermostat to adjust the temperature setpoint is more impactful to the building occupants than simply reading the value of a temperature sensor).

To account for the fact that some equipment and points are more sensitive than others, we used the Analytic Hierarchy Process (AHP) [47] to create a suitable scheme for weighting the sensitivity of each type of equipment and sensing/control points. AHP is a decision-making technique that can be used to prioritize the attributes relevant to a decision-making task: by pairwise comparing these attributes, it helps stakeholders decide on the importance of each attribute relative to others. It has been used for a wide variety of applications, including assessing risk in operating pipelines [19] and quantifying the overall quality of software systems [35]. In principle, this process should be undertaken by facility-management personnel in collaboration with building owners and occupants. For this work, we answered the questions specified in Appendix A.2.1 on the scale defined in Appendix A.2.2 to develop a list of weights for all types of points (sensors and setpoints) in our model. These weights are shown in Table 4.2.

In addition to its type, the sensitivity cost of a particular piece of equipment or a sensing/control point depends on what other, and how many, physical components they could impact. This intuition is illustrated in Figure 2.1.

Table 4.2: Weights of the two types of points modeled: sensors and setpoints.

Sensors	**Weight**
Temperature_Sensor	0.347
Damper_Position_Sensor	0.204
Occupancy_Sensor	0.246
Humidity_Sensor	0.204
Setpoints	
Temperature_Setpoint	0.413
Humidity_Setpoint	0.260
Air_Flow_Setpoint	0.328

In this figure, *Temperature_Setpoint_1-12* is a *Point* element of type *Setpoint* and *Temperature_Sensor_1-12* is a *Point* element of type *Temperature_Sensor*. These are points of the *VAV_1-12 Equipment* and are both located in *Room_1-1-121*. *VAV_1-12* feeds fresh air into *HVAC Zone_1-12* which is of type *Location*. *Room_1-1-120* and *Room_1-1-121*, which are also of type *Location*, are parts of the *HVAC Zone_1-12*. In estimating the sensitivity of this room, we argue that one would have to take into account the fact that VAV equipment can be impacted by the actions of the room occupants, who may read the *Temperature_Sensor_1-12* value and control the *Temperature_Setpoint_1-12*. Thus, *Room_1-1-121* is more sensitive than *Room_1-1-120* and occupants need to be aware of this when granting permissions to other occupants and visitors.

4.2.3 Defining Sensitivity of an Indoor Path

Finally, the overall sensitivity cost of a potential indoor *path* is calculated as the sum of all the costs of the rooms r it includes based on their security zone classification, plus the costs of all points p located in the rooms r. The total cost of each point p is initially assumed to be the value of 1, and increases with the number of rooms r affected by point p. Furthermore, the cost is scaled by weight(p), the sensitivity of the point as established through the AHP process and seen in Table 4.2. This intuition is captured by the following function:

$$\text{cost}(path) = \sum_{r \in path} \text{cost}(r) \tag{4.1}$$

$$= \sum_{r \in path} \left(\text{sensitivity}(r) + \sum_{p \text{ hasLocation } r} \text{weight}(p) \times (1 + control(p)) \right), \tag{4.2}$$

where *path* is a sequence of rooms r, sensitivity(r) is the numerical value of security zone classification for room r, p is a point (e.g., a setpoint or a sensor) that is part of a subsystem, weight(p) is the weight given to point p (determined by AHP), and control(p) is the number of locations or zones affected by point p. For example, in Figure 2.1, we see that *Temperature_Setpoint_1-12* has control(p) = 2 as ambient air temperature of two rooms would be affected by adjusting this setpoint. A complete calculation for the costs of *Pathway 1* and *Pathway 2* can be found in Appendix A.1.4.

It is important to explain here the role of the building model in computing the path-sensitivity cost function above. We have developed a set of SPARQL queries based on the building's RDF model (see sub-graph in Appendix A.1.1) to compute:

- the sequence of adjacent *rooms* leading from one location to another, e.g., the main building door to the meeting room; and
- the set of *points* located in a room, their types, and the locations they influence through control.

We seek to answer the following questions with SPARQL queries in the namespaces found in Appendix A.1.2:

Q1 – what are the possible sequences of adjacent resources (locations or doors), starting at *main entrance* and reaching a specific *meeting room*?

```
1  SELECT ?element
2  WHERE { ?element bot:adjacentElement <Resource>. }
3
4  SELECT ?element # reverse query of lines 1-2
5  WHERE { <Resource> bot:adjacentElement ?element. }
```

The two queries above are repeatedly invoked, starting with the building's *main entrance* as $< Resource >$ to find an adjacent door or room $?element$ using the query from lines 1-2. For each of the $?elements$, if $?element$ is a door, then the query from lines 3-4 is executed next using $?element$ as $< Resource >$ to find more doors or rooms. Otherwise, the query from lines 1-2 are repeated with the $?element$ as the new $< Resource >$. This process is repeated until the destination *meeting room* $< Resource >$ has been reached. Note that when a query returns multiple rooms or doors, we need to repeat this process for each of them separately. Using these two queries allows the Path-Planning Service to perform a recursive depth-first search by finding adjacent resources to each 'Room'.

Q2 – what is the sensitivity of a *location* based on its *security zone* classification?

```
SELECT ?seczone
WHERE {
    ?seczone bot:hasSpace <location> .
}
```

This query helps to determine the *sensitivity* of $< location >$ where $?seczone$ is subsequently translated into a number based on its ordinal property described in Section 4.2.1.

Q3 – what *points* are in a *location*? What kinds of *points* are they? Can the points influence other locations through *controls* relationship?

```
1  SELECT ?point ?subsubtype ?subtype
2  WHERE {
3      <location> bf:isLocationOf ?point .
4      ?point rdf:type ?type .
5      ?type rdfs:subClassOf* ?subsubtype .
6      ?subsubtype rdfs:subClassOf ?subtype .
7      ?subtype rdfs:subClassOf brick:Point .
8  }
9
10 SELECT ?point ?location
11 WHERE {
12     <location> bf:isLocationOf ?point .
13     ?point bf:controls ?command .
14     ?equipment bf:hasPoint ?command .
15     ?equipment bf:feeds ?zone .
16     ?zone bf:hasPart ?location .
17 }
```

The first query from lines 1-7 determines the points that are located in $< location >$ and their types (subsubtype = Temperature_Setpoint and subtype = Setpoint). The second query from lines 9-16 decides whether the point influences any other locations by determining the equipment of the command that the point controls and seeing if the equipment feeds any locations. These queries are executed by the Building-Representation Service to determine the cost of an indoor-path.

4.3 Summary

In this chapter, we discussed the design and implementation of the commercial building simulator. The interaction between the simulator, Path-planning service and Access-control service were described along with a Calendar application that serves as the user interface for meeting hosts. In the next chapter, we talk about the usage of the simulator for evaluating the performance of the proposed access control system.

Chapter 5

Performance Evaluation through Simulation

In this chapter, we describe in detail the various components of a building occupancy simulator and their interactions with Path-Planning and Access-Control services. The simulator synthesizes movement of people in a building and interacts with these services in order to perform a comprehensive analysis of the proposed solution. We then assess the performance of the proposed access management system using the commercial building simulator. We compare the latency of each kind of transaction which manage entities and access rules in the smart contracts. To measure the scalability of the solution, we compare the throughput of the Ethereum network as we increase the number of miner nodes.

5.1 Simulating Occupant Presence and Movements

The architecture in Figure 5.1 depicts the interaction of the simulator, services and the user interface. As shown in this figure, the simulator takes a set of input parameters from the initiating application which could be the front end calendar application or a command line prompt. The set of inputs include the number of days for which the occupant movements need to be simulated, the duration of the meetings, the type of day (Busy, Average or Quiet) and a list of meeting hosts and meeting participants. For instance, the simulator is asked to synthesize data for one quiet day and for meeting duration ranging from 60 minutes to 150 minutes. The simulator assumes that meeting periods are rounded off to the nearest 30 minutes. By default, it creates schedules starting from 08:00 hours to 17:00 hours, which can be changed by the initiator. The set of conference rooms and designated meeting locations can be provided as input or derived from the Path-Planning Service of the building. For the purpose of simulation, a set of participants is required as input to the simulator.

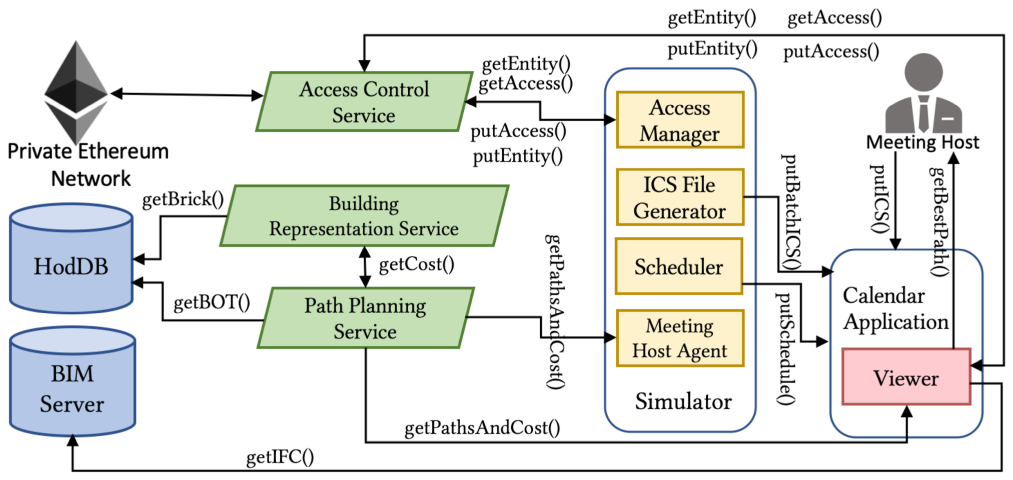


Figure 5.1: Architecture of Simulator's interaction with Access Control service and Path Planning service

5.2 Simulating Occupant Arrivals and Meetings

The component called Scheduler, depicted in Figure 5.1, is present inside the simulator which is responsible for using the above described inputs to create a meeting schedule for the given type of day. We model the number of meetings held throughout the day in all conference rooms by a Poisson process which is used to model arrivals and services in a queuing system. Poisson arrival process implies that the inter-arrival times are exponentially distributed. Exponential distribution is memory-less, so the time until the next arrival does not depend on the time spent after the previous arrival.

Consider a stochastic process $T = \{T_n, n \geq 1\}$ which records the occurrence time of the n^{th} event in some experiment. For any interval $[s, t)$ we define the random variable

$$N([s,t)) = \sum_{n \geq 1} 1(T_n \in [s,t)) \tag{5.1}$$

which represents the number of occurrences of the point process T in $[s, t)$. We say $N([s, t))$ is a Poisson process with rate λ for $0 \leq s < t$ if

- $\tau_n = T_{n+1} - T_n$ is a collection of independent and identically distributed (iid) random variables
- τ_n is exponentially distributed with rate $\lambda > 0$

Algorithm 1: Simulate meetings(type_of_day, sim_length)

```
Result: $(loc_n, s_n, e_n, A_n, D_n)$
$\lambda \leftarrow MeetingRate(type\_of\_day)$;
$a, b \leftarrow MeetingLength(type\_of\_day)$;
$\mu \leftarrow ArrivalRate(type\_of\_day)$;
while $\tau < sim\_length$ do
    $s_n \sim Exp(\lambda)$;
    $\tau \leftarrow \tau + s_n$;
    $\ell \sim U(a, b)$;
    $e_n \leftarrow \ell + s_n$ ;
    $loc_n \leftarrow rand(locations, s_n, e_n)$ ;
    $A_n \leftarrow SimulateArrivalTimes(s_n, \mu)$ ;
    $D_n \leftarrow SimulateDepartureTimes(e_n, \mu)$ ;
    $store(loc_n, s_n, e_n, A_n, D_n)$ ;
end
```

Initially, we assume that the total number of meetings per day is a Possion process. The meeting rate for this Possion process, λ, is dependant on the *type_of_day*, which decides the load of the system (busy, average or quiet). The start time of the next meeting is sampled from an exponential random variable and is added to the start time of the previous meeting which was held in any one of the conference rooms in the building. The duration of the next meeting is sampled from a uniform distribution. The conference room for the next meeting is randomly selected from the *available* conference rooms. This process is described in Algorithm 1.

We derive the duration of meetings from the function MeetingLength($type\_of\_day$), which uses a uniform distribution and samples the length of each meeting from this distribution. We consider a window of 30 minutes before the meeting starts and after it ends, and use a Poisson process to simulate arrivals and departures in these two windows as described in Algorithm 2 and Algorithm 3. The number of participants are sampled from another uniform distribution, which depends on the $type\_of\_day$. The rate of arrival, μ is computed by the ArrivalRate($type\_of\_day$) function in Algorithm 1. The inter-arrival times are sampled from the exponential distribution with the mean being μ.

Algorithm 2: SimulateArrivalTimes(s$_n$)

```
Result: Arrival times of participants in $n^{th}$ meeting
$\kappa \leftarrow s_n - 30minutes$;
while $\kappa < s_n$ do
    $a_n \sim Exp(\mu)$;
    $\kappa \leftarrow \kappa + a_n$;
    $store(a_n)$;
end
```

Algorithm 3: SimulateDepartureTimes(e$_n$)

```
Result: Departure times of participants in $n^{th}$ meeting
$\kappa \leftarrow e_n + 30minutes$;
while $\kappa > e_n$ do
    $d_n \sim Exp(\mu)$;
    $\kappa \leftarrow \kappa - d_n$;
    $store(d_n)$;
end
```

Once the meeting schedule is generated, the simulator returns the meeting schedule, arrival and departure times of the participants for all the meetings to the initiator (the Calendar application or the user). The ICS File Generator component of the simulator in Figure 5.1 uses this schedule to generate ICS Calendar files for each meeting. These files store information of the host, attendees, meeting location, date and duration of the meeting. The next

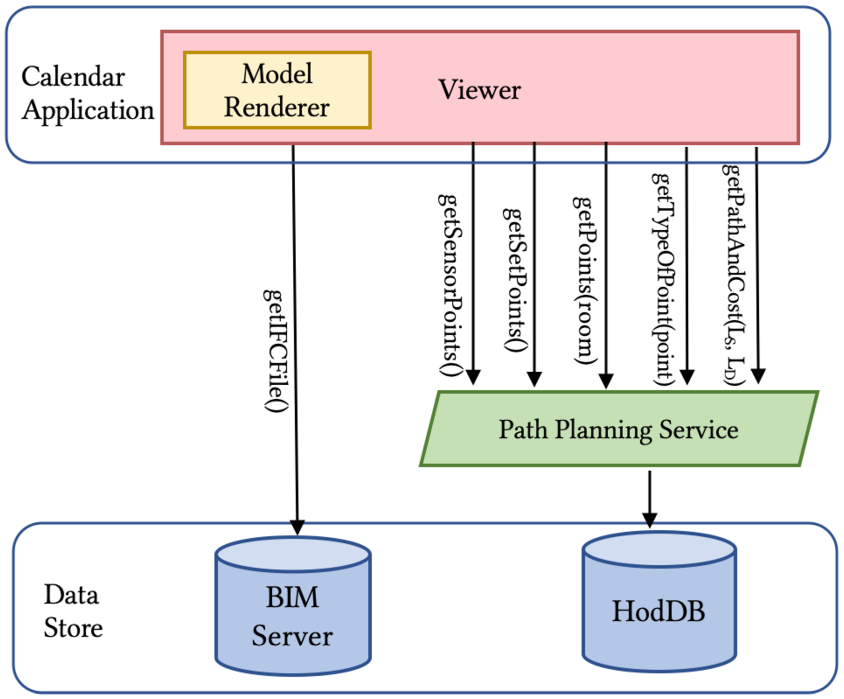


Figure 5.2: Interactions between Calendar application's Viewer and Path-Planning Service.

step could be done in two ways- the simulator could use the schedule to run synthetic path planning and access management or the Calendar application can take these files and extracts the meeting information, as described in Section 5.2.1, to run a manual simulation using the batch of ICS files.

In the automated process, the simulator calls the Path-Planning Service to provide all possible paths between any building entrance and the specified meeting location, along with the costs of permitting visitors take these paths (getPathAndCost() in Figure 5.1). The component *Meeting host agent* of the simulator in Figure 5.1 represents a modelled agent for the meeting host. The agent selects the best path for the meeting participants of the meeting, which, by default, is the minimum cost path. The simulator now takes this path information and sends requests to the access control service to delegate access to each meeting participant as described in Section 2.2.

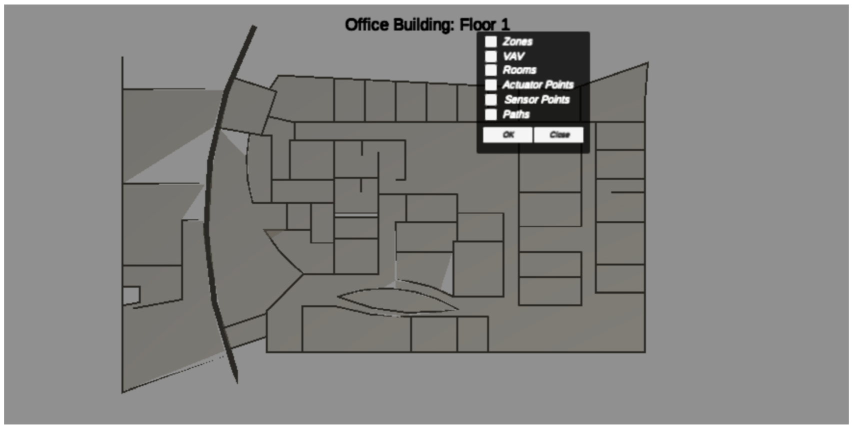


Figure 5.3: Calendar application's menu options to reason about points and spaces.

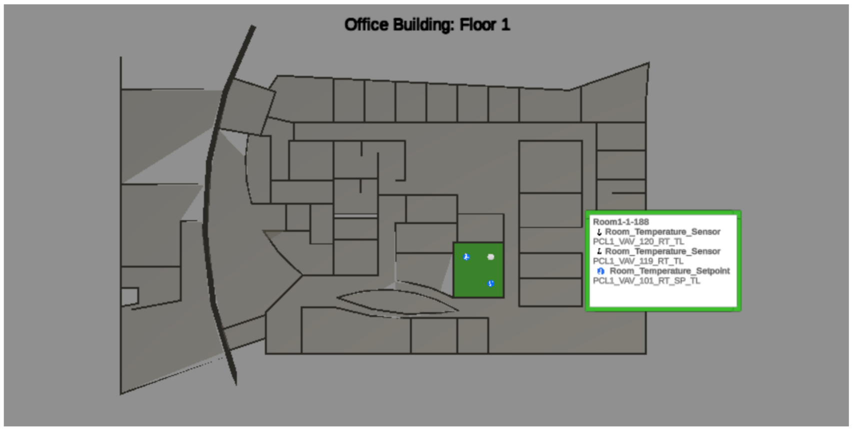


Figure 5.4: Calendar application's viewer highlighting a room and points present in the room.

The simulator simultaneously runs to send access verification requests to the access control service to synthesize the entrance of people into the building and rooms in their path according to the simulated arrival times. The simulator then records response times for each request and also maintains the occupancy information of the building.

5.2.1 Calendar Application

Upon receiving the ics Calendar files from the meeting host manually or from the simulator, the Calendar application extracts the list of meeting participants and meeting location. Firstly, the Calendar application receives the IFC

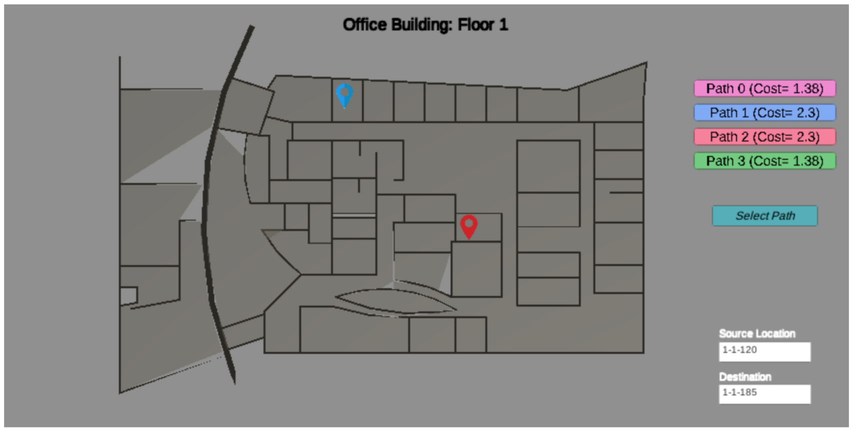


Figure 5.5: Viewer displaying possible path options between selected source and destination locations.

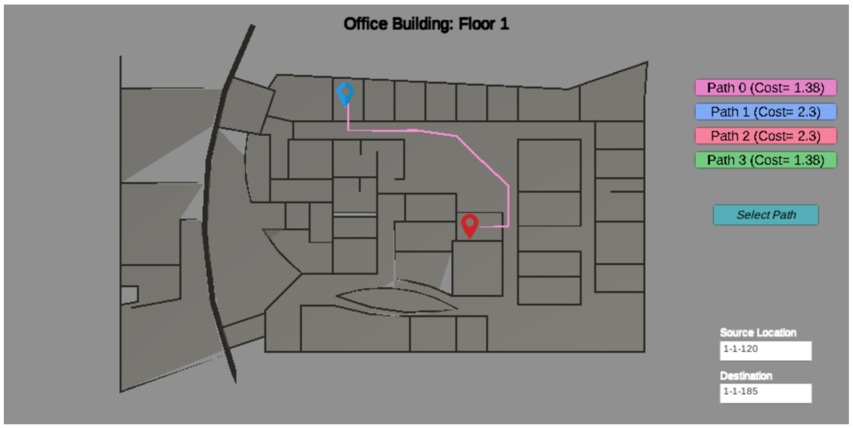


Figure 5.6: Meeting host interactively selects the best cost path for the meeting participants.

file of the building from a BIM Server. This file contains the physical layout of the building and is rendered into a 3D floor plan using the Model renderer component in Figure 5.2. The Viewer displays the 3D floor plan of the building which helps the host interact with the model, as in Figure 5.3. If a meeting location is detected inside the ics file it is stored as the destination location. Otherwise, the Calendar application waits for the meeting host to select a meeting location as the destination and a source location (any entrance of the building for visitors and an office location in case of an occupant of the building). Upon receiving the source and destination, the Path-Planning Service is requested to provide all the paths between these locations along with their respective costs. The received paths are processed and displayed according to their costs on the viewer for the meeting host, as in Figure 5.5 and Figure 5.6, through which the best fit path for the meeting participants can be selected. We explain how paths are selected in Section 4.2.

The rendered model can be used to view and reason about various sensors and actuators present on the floor, for instance the relationship between an actuator and its linked sensor along with their physical location on the floor. The model can also show the points present inside a selected room and the detailed information about each point that is highlighted, as in Figure 5.4. All this is possible with the interaction between the Calendar application and the Path-Planning Service, as described later in Section 4.2.

5.3 Simulation of meetings and people in the building

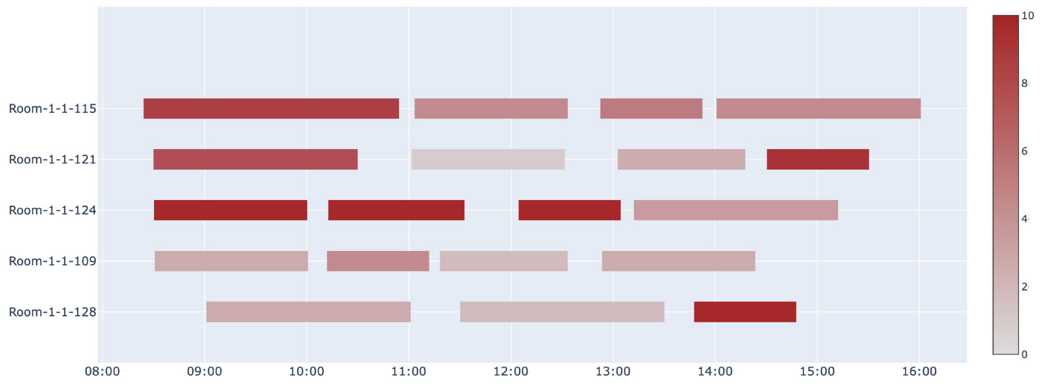
We simulate a number of concurrent meetings in the example building depicted in Figure 4.1 to evaluate the performance of the proposed system in realistic scenarios. This building has five conference rooms that could be used to hold meetings in parallel. To create different load levels, we consider 3 types of days: busy, average, and quiet days as described in Chapter 4. We assume that there are more concurrent meetings with more participants on a busy day than an average day or a quiet day. For each type of day, we model the number of meetings held in each conference room by a Poisson process. Hence, the intervals between successive meetings held in the same conference room are exponentially distributed.

For example, we assume that n number of meetings take place on an average day. The value of n can be derived from a uniform distribution. The meeting rate λ in Algorithm 1 is calculated by dividing the number of meetings in a day by the duration over which these meetings can span. For a typical commercial building, the working hours are between 8 to 9 hours. Now, consider a meeting scheduled between 14:00 to 15:00 with 7 participants. We assume that the duration of a meetings could span from 60 minutes to 150 minutes, which is assigned by the function MeetingLength($type\_of\_day$). The meeting length is sampled from a uniform distribution bound by 60 and 150 minutes.

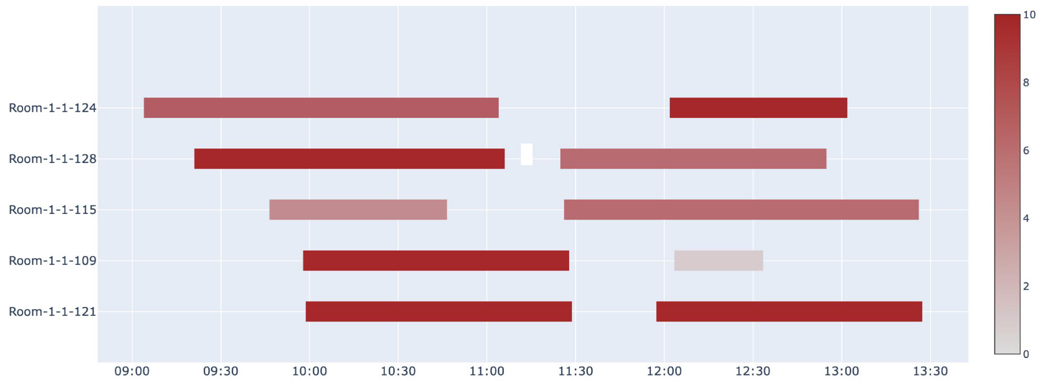
The number of participants are sampled from a uniform distribution, in this case there are 7 participants. These participants arrive any time 30 minutes before the meeting, thus the rate μ here is 7 participants divided by 30 minutes, which is $\mu = \frac{7}{30}$. The inter-arrival times for this meeting are calculated in Algorithm 2 and Algorithm 3 which are sampled from the exponential distribution with the mean being μ. The resultant arrival instances would be between 13:30 and 14:00 and departure instances would be between 15:00 and 15:30.

Figure 5.7 shows example schedules for all meeting rooms in the 3 types of days we considered. The color intensity depicts the number of participants of each meeting; the darker the color is the more people attend the meeting. Figure 5.8 shows the cumulative number of people in the building throughout the day for the 3 types of days. We have a maximum of 70 meeting participants on busy and average days, and a maximum of 30 meeting participants on quite days. There are no occupants in the building before 08:00 and after 17:00.

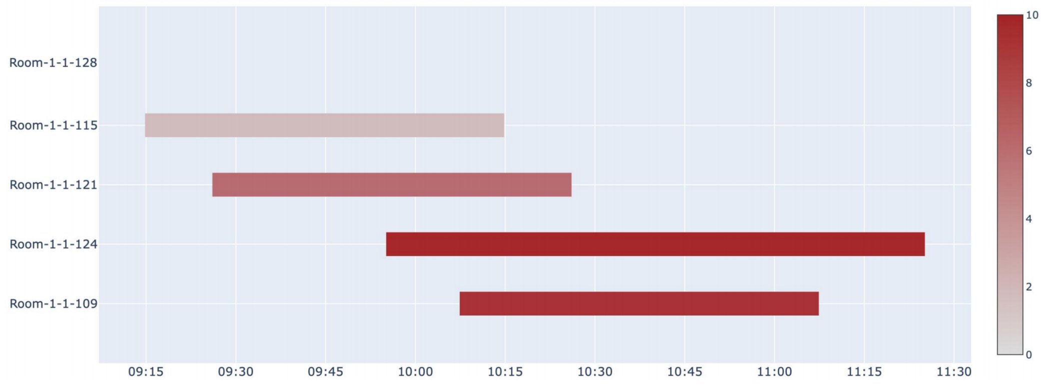
When access is delegated to an attendee, the paths to these meeting rooms are received from the path planning service. Accordingly, we calculate the number of requests a participant makes to receive and verify access to attend one meeting. The average number of requests is between 4.5 and 5.5 requests



(a) Busy Day Schedule



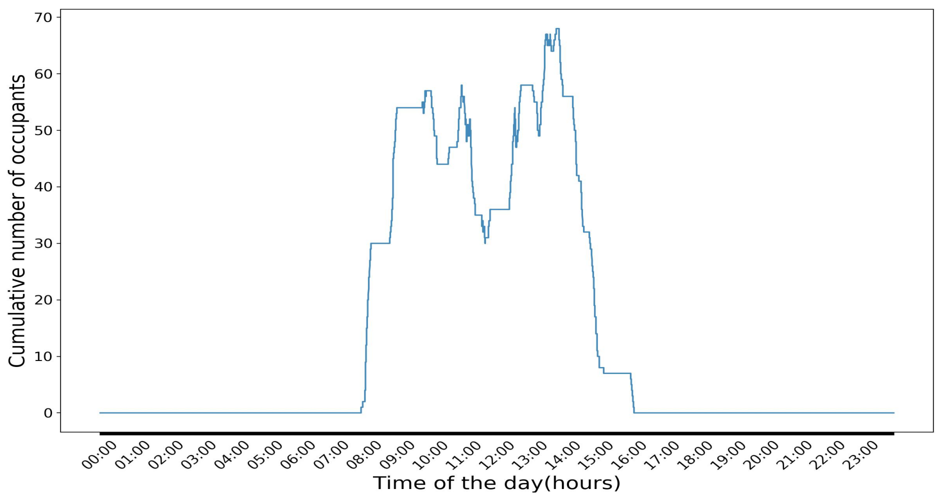
(b) Average Day Schedule



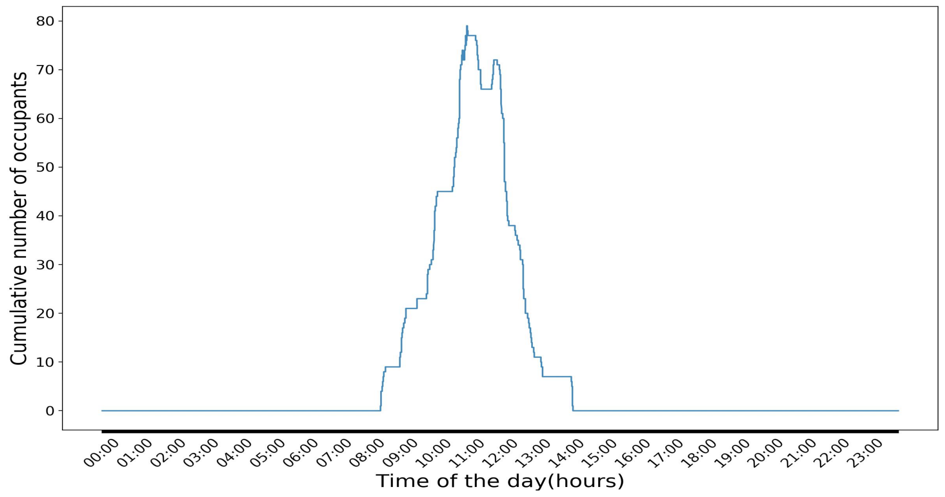
(c) Quiet Day Schedule

Figure 5.7: Meeting schedules in 3 different types of days

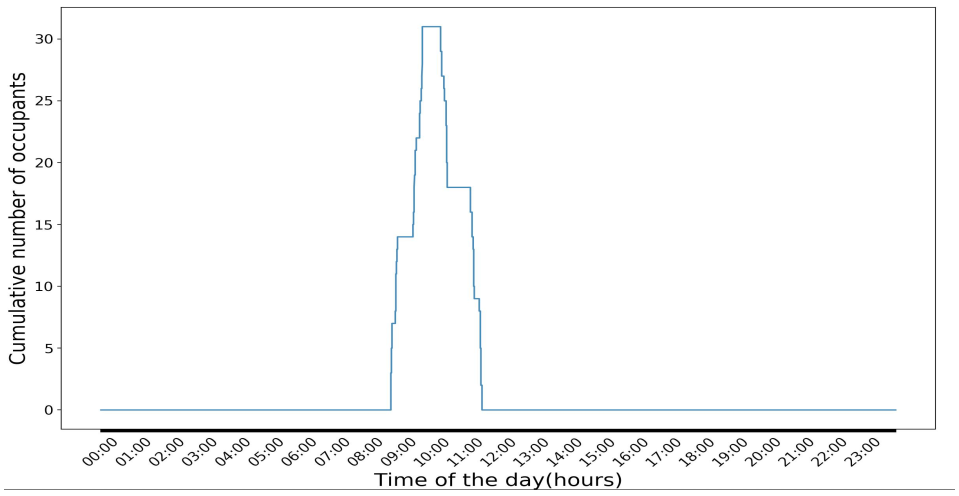
per participant in our simulations.



(a) Busy Day Occupancy



(b) Average Day Occupancy



(c) Quiet Day Occupancy

Figure 5.8: Occupants in the building during 3 different schedules

5.4 Delay and cost analysis of Managing Entities and Access Rules

Performance evaluation is done on a private Ethereum network consisting of 2 mining nodes. The API sends simultaneous and parallel requests to these nodes and transactions are executed by the miner nodes. We measure the latency of transactions as perceived by the user, i.e., the time between the user issuing a request to the access control service and receiving a response from the service. As described in Section 2.2, the access control service interacts with the blockchain mining nodes to issue transactions. Entities are created and access rules are added in an offline fashion, usually well before the meeting start time. Table 5.1 describes the average delay for adding a new entity for a user unknown to the system and adding access rules for each participant of the meetings to the blockchain. It also shows the cost associated with transactions to add new entities and access rules. At the time of evaluation, one ether was equal to approximately 143.68 USD.

Table 5.1: Summary of transaction delays and costs.

Type of day	**avg. delay of adding entity**	**avg. delay of adding access rule**	**avg. cost for adding entity**	**avg. cost for adding access rule**
Busy Day	0.76 seconds	17.3 seconds	0.019 USD	0.0044 USD
Average Day	1.2 seconds	14.3 seconds	0.019 USD	0.0045 USD
Quiet Day	1.16 seconds	12.8 seconds	0.019 USD	0.0044 USD

We now present the distribution of delays obtained for verifying user's access privileges, e.g., when meeting participants arrive to the building. Our simulations indicate that this type of request is completed between the range of 0.26 and 0.37 seconds for each type of day. Thus, we can expect a maximum delay of 0.37 seconds to decide whether or not to grant an access request. This delay is within the acceptable range in real applications.

5.4.1 Throughput and Latency of Verifying Entities and Access Rules

Lastly, we increase the number of mining nodes in the private blockchain network from 1 up to 4. We consider two performance metrics, namely throughput and average latency, and we measure performance as the system load (the total number of transactions) increases in each case. We assume that transactions are evenly distributed among the nodes in the network. Figure 5.9 shows the performance evaluation results obtained from 5 independent runs for an average day schedule. It can be seen from the figure that (a) the average latency decreases in most cases as we increase the offered load, and (b) throughput increases linearly with the offered load until it reaches a maximum, which depends on the number of nodes. When the load exceeds this threshold, performance starts to fall apart. Observe that increasing the number of nodes improves performance in terms of both throughput and latency in general. With 4 nodes in the blockchain network, according to Figure 5.9, we can handle around 2400 verification requests per minute with an average latency as low as 30 milliseconds. From Table 5.1 and Figure 5.9, we can say that verifying presence of entities and validity of access can be done much faster than adding new entities and access rules.

5.4.2 Practical Considerations for Using The Blockchain

Privacy: Maintaining privacy on blockchain is a complicated issue because transactions and user's balances in a blockchain are open to public viewing. There are several approaches to address privacy concerns in a blockchain based solution. For example, secure multi-party computation splits the smart contract between a number of parties with secret keys to compute parts of the smart contract so that a complete picture of a smart contract is not given. Zero-knowledge proofs can provide verification of smart contracts without revealing any information except for the proof to be true; this process can be quite costly. Commitment schemes allow for proofs to be verified with minimal disclosure of secrets. Mixing is also an option where transactions are hidden by

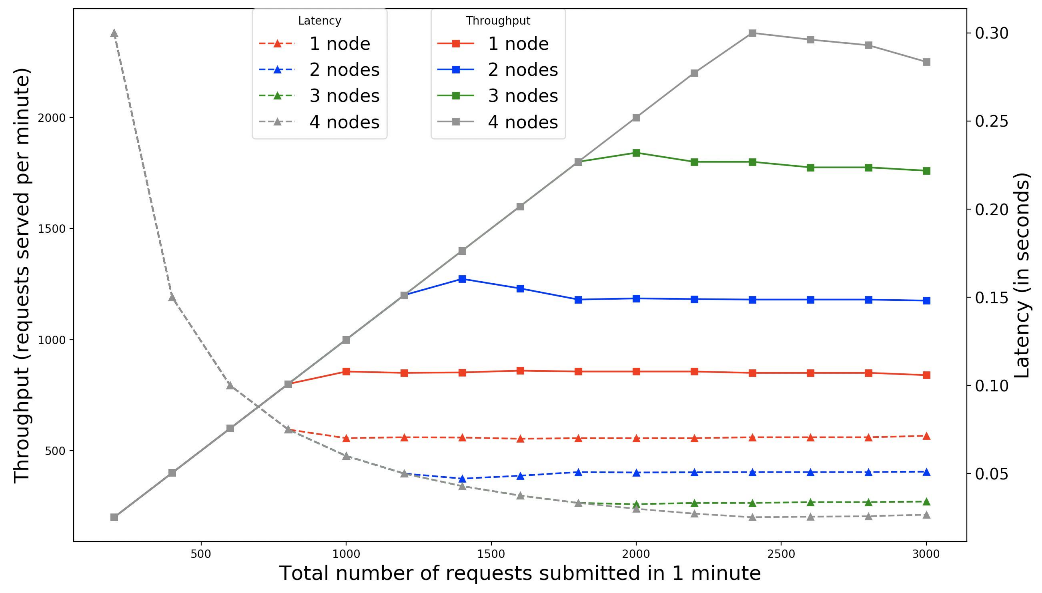


Figure 5.9: Comparison of throughput and latency for different numbers of nodes and amounts of offered load

generating additional transactions to create noise and hide the original transaction. Furthermore, user privacy can also be maintained in a hybrid blockchain solution where identity is managed by an external public blockchain service, while access smart contracts are maintained on a private blockchain. Our proof-of-concept implementation uses a private Ethereum network which addresses privacy concerns to some extent as all participating nodes are within the organization. Nevertheless, any of the above approaches can be implemented on top of our access-control service when the meeting participants and times are sensitive and must be protected from some nodes in the network.

Transaction fees: Cryptocurrency fees are a fundamental part of blockchain-based software platforms. In some public ledgers there is a minimum fee required for a transaction to be accepted, which helps avoid unwanted and inappropriate transactions. In this work, the creation of entities and access rules require a fee, which has to be paid by the entity initiating and signing the transaction. However, checking access privileges does not cost transaction fees, which constitute the most common type of operations. When a user tries to access a resource, the API is called to query information from the smart contract and decide whether the requested access is allowed. This can be done from any resource and does not incur any fee. It should also be noted that with a private Ethereum network, the difficulty of the mining process can be decided by the management, and ether could be mined and potentially transferred to the accounts of participating nodes as needed.

Block time: Transactions take time to get accepted into the blockchain. However, verifying access privileges does not need to execute transactions to query information from the smart contract's data. The information can be retrieved immediately from the blockchain by the devices when the API is called. Apart from this, the creation of new entities and access rules can have long delays in execution. This implies that users may have to wait for some time till their access privileges are granted, since this process requires the execution of two transactions, namely (a) the creation of a new entity and (b) the creation of a new access rule. To mitigate the potentially long wait times, one might raise the transaction fees for the creation and revocation operations to minimize the time spent adding or removing the entities and access rules into the blockchain network [42].

5.5 Summary

In this chapter, we described how the commercial building simulator is used for performance evaluation. With an example *type_of_day*, we described how the rate of meetings and inter-arrival times are calculated to build a schedule for one day. This schedule was used to make requests to the Access-control service to evaluate the throughput and latency of the transactions made on the private Ethereum network. Lastly, we discuss some measures to be considered for good performance of the proposed system in practical scenarios.

Chapter 6

Conclusion

In this thesis we proposed a methodology that supports reasoning about and flexibly managing the access privileges of occupants and visitors in multi-tenant commercial buildings. Controlling access to these areas and other spaces within the building is a complex problem. Our method uses smart contracts to manage the space and equipment access privileges of users, at specific times and subject to specific constraints.

The current practice of using a centralized infrastructure relies on prox cards for occupants and requires one of these occupants (or dedicated security personnel) to escort visitors to their meeting locations using their own prox cards. This process is onerous and costly, and in fact does not provide any kind of real access to visitors.

A private blockchain uses distributed ledger technology enabling tenant organizations to easily read, write, and audit transactions. Additionally, since there is no single trusted entity and no single point of failure, our access control solution (based on blockchain) has better availability and reliability compared to a centralized solution running on a single server. Our solution introduces time-restricted building passes and uses a decentralized access control system, relying on building models, to reduce the cost of access control in a large multi-tenant building and make it more flexible (allowing the update of passes) and secure (by specifying a specific path that must be taken to the meeting location). The operational cost of our access control solution is much lower than the current best practice.

Furthermore, our methodology combines the Brick and BOT models of a building to plan an indoor path between two locations and to determine the path's "cost" in terms of the sensitivity of the spaces it goes through and the equipment contained in these spaces. The unification of RDF graphs that portray building metadata and smart contracts for authorization is done through the use of API endpoints. We demonstrated through an example use case that the proposed access-control methodology is suitable for managing access privileges in large multi-tenant commercial buildings and can greatly simplify the labor-intensive security protocol that is currently being followed in such buildings. Through our simulations, we inferred that the request to verify access for a person trying to unlock a door is completed between the range of **0.26 and 0.37 seconds** with just 2 mining nodes, which is typically within the acceptable range in real applications.

To evaluate the overall performance of the proposed solution, we design and implement a simulator that synthesizes occupant presence and movements in the building using scheduled meetings fabricated by a Poisson arrival process. We used the data generated by the simulator to perform load testing on the blockchain network to assess its throughput and latency.

We believe that the proposed solution is suitable for access control in large multi-tenant commercial buildings, greatly simplifies the labor-intensive security protocol that is currently being followed in such buildings, and offers advantages over a centralized access control system in this context. However, the blockchain-based access control solution which we proposed in this work may not be better than a centralized solution for all types of buildings. Depending on the size of the building and the number of visitors that come to the building on a day to day basis, the number of state-change transactions would vary significantly, effecting the performance of the proposed solution. We performed the cost analysis for busy, average and quiet days in our select building using a proportional estimate of the number of transactions.

The computational requirements for the proposed solution are higher than the centralized infrastructure as it requires multiple dedicated blockchain nodes. On top of this, the Brick and BOT models for each building are required in order to use the proposed solution for any commercial building, limiting the adaptability of this solution.

Future work would include investigating methods to reduce the cost (ether) of using smart contracts. With small modifications to interact with the actual building's BMS, the system can be extended to enable real-world applications without changing the authorizing smart contracts. A comparative study of using data from BMS and smart contracts directly would be performed in terms of cost and time.

Furthermore, incorporating the use of roles instead of individual users receiving access is another extension that can be made in future work. Using role-based access control could help group similar types of users into one role and provide them with identical access. For example, all the occupants of an office room with multiple workstations should have access to the same set of physical resources. This approach would be applicable in single-tenant buildings.

Apart from authorization, authentication, and revocation, it is possible to build several other applications, like occupancy monitoring, localization, or path planning using accessibility information of the building, on top of the existing authorizing smart contracts and the building metadata schema, thanks to adaptability and flexibility of the proposed solution.

To reiterate, the contributions of this thesis are:

- We develop a smart-contract based solution to flexibly manage access control for large commercial buildings. Distributed services are built on top of the deployed smart contracts to manage multi-occupancy buildings. The access delegation is auditable with the usage of blockchain, which is a scalable, trustless, peer-to-peer solution that operates transparently.

- We build a simulator that generates data to synthesize movement of people in a large commercial building. The simulator uses a unified building model created by aligning BOT and Brick ontologies. This model supports path planning and uses an external cost function to prioritize paths. Building usage and access delegation data is generated by the simulator for a specified duration of time.

- We evaluate the proposed access management solution using the simulator.

References

[1] Project haystack. `http://project-haystack.org/`.

[2] Ethereum foundation, solidity documentation release 0.5.3. *–URL: https://solidity.readthedocs.io/en/v0.5.3/*.

[3] M. P. Andersen et al. WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts. Technical Report UCB/EECS-2017-234, EECS Department, University of California, Berkeley, Dec 2017. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-234.html`.

[4] M. P. Andersen et al. Democratizing authority in the built environment. *ACM Transactions on Sensor Networks (TOSN)*, 14(3-4):17, 2018.

[5] M. P. Andersen et al. Wave: A decentralized authorization framework with transitive delegation. In *28th USENIX Security Symposium*, pages 1375–1392. USENIX Association, 2019.

[6] O. Ardakanian, A. Bhattacharya, and D. Culler. Non-intrusive occupancy monitoring for energy conservation in commercial buildings. *Energy and Buildings*, 179:311–323, 2018.

[7] B. Balaji et al. Brick: Metadata schema for portable smart building applications. *Applied Energy*, 226:1273–1292, 2018.

[8] V. Bazjanac and D. Crawley. Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings. In *Proceedings of Building Simulation*, volume 2, pages 661–667, 1999.

[9] D. Beckett. Turtle-terse rdf triple language. *http://www. ilrt. bris. ac. uk/discovery/2004/01/turtle/*, 2008.

[10] J. Bernal Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, R. Torres Moreno, and A. Skarmeta. Privacy-preserving solutions for blockchain: Review and challenges. *IEEE Access*, 7:164908–164940, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2950872.

[11] K. Z. Bijon, R. Krishnan, and R. Sandhu. A framework for risk-aware role based access control. In *2013 IEEE Conference on Communications and Network Security (CNS)*, pages 462–469. IEEE, 2013.

[12] V. Buterin. A next-generation smart contract and decentralized application platform. *White paper*, pages 1–37, 2014.

[13] J. Callas et al. OpenPGP message format. Technical report, 2007.

[14] G. Caronni. Walking the web of trust. In *Proceedings of the 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 153–158. IEEE, 2000.

[15] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[16] L. Chen and J. Crampton. Risk-aware role-based access control. In *International Workshop on Security and Trust Management*, pages 140–156. Springer, 2011.

[17] J. P. Cruz, Y. Kaji, and N. Yanai. RBAC-SC: Role-based access control using smart contract. *IEEE Access*, 6:12240–12251, 2018.

[18] S. Dawson-Haggerty et al. BOSS: Building operating system services. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*, pages 443–457. USENIX, 2013.

[19] P. K. Dey. Analytic hierarchy process analyzes risk of operating cross-country petroleum pipelines in india. *Natural Hazards Review*, 4(4):213–221, 2003. doi: 10.1061/(ASCE)1527-6988(2003)4:4(213).

[20] A. Dmitrienko, A.-R. Sadeghi, S. Tamrakar, and C. Wachsmann. Smart-tokens: Delegable access control with NFC-enabled smartphones. In *International Conference on Trust and Trustworthy Computing*, pages 219–238. Springer, 2012.

[21] A. Dorri, S. S. Kanhere, and R. Jurdak. Towards an optimized blockchain for iot. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 173–178. IEEE, 2017.

[22] J. R. Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[23] D. Ferraiolo, J. Cugini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th annual computer security application conference*, pages 241–48, 1995.

[24] G. Fierro and D. E. Culler. Design and analysis of a query processor for brick. *ACM Transactions on Sensor Networks (TOSN)*, 14(3-4):1–25, 2018.

[25] A. Hamieh, A. B. Makhlouf, B. Louhichi, and D. Deneux. A bim-based method to plan indoor paths. *Automation in Construction*, 113:103120, 2020.

[26] A. Helal, K. Cho, W. Lee, Y. Sung, J. Lee, and E. Kim. 3d modeling and simulation of human activities in smart spaces. In *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 112–119. IEEE, 2012.

[27] J. Howell. Number of connected iot devices will surge to 125 billion by 2030, ihs markit says. *IHS Markit Technology, Oct*, 24, 2017.

[28] V. C. Hu et al. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800(162), 2014.

[29] V. Jacobson et al. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 1–12. ACM, 2009.

[30] A. A. E. Kalam et al. Organization based access control. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks*, pages 120–131. IEEE, 2003.

[31] S. Kandala, R. Sandhu, and V. Bhamidipati. An attribute based framework for risk-adaptive access control models. In *Proceedings of the 6th International Conference on Availability, Reliability and Security*, pages 236–241. IEEE, 2011.

[32] M. A. Khan and K. Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[33] T. H. Kolbe, G. Gröger, and L. Plümer. Citygml: Interoperable access to 3d city models. In *Geo-information for disaster management*, pages 883–899. Springer, 2005.

[34] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Symposium on Security and Privacy (SP)*, pages 839–858. IEEE, 2016.

[35] P. Kumar and S. K. Singh. A comprehensive evaluation of aspect-oriented software quality (AOSQ) model using analytic hierarchy process (AHP) technique. In *Proceedings of the 2nd International Conference on Advances in Computing, Communication, Automation*, pages 1–7, Sep. 2016.

[36] T. Le and M. W. Mutka. Access control with delegation for smart home applications. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI)*, pages 142–147. ACM, 2019.

[37] K.-J. Li. Indoorgml–a standard for indoor spatial modeling. *The Inter-*

national Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 41:701, 2016.

[38] Y.-H. Lin, Y.-S. Liu, G. Gao, X.-G. Han, C.-Y. Lai, and M. Gu. The ifc-based path planning for 3d indoor spaces. *Advanced Engineering Informatics*, 27(2):189–205, 2013.

[39] S. Nakamoto and A. Bitcoin. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 2008.

[40] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, 2016.

[41] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications magazine*, 32(9):33–38, 1994.

[42] O. Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5(2):1184–1195, 2018.

[43] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman. FairAccess: a new Blockchain-based access control framework for the Internet of Things. *Security and Communication Networks*, 9(18):5943–5964, 2016.

[44] A. Ouaddah, A. Abou Elkalam, and A. A. Ouahman. Towards a novel privacy-preserving access control model based on blockchain technology in iot. In *Europe and MENA cooperation advances in information and communication technologies*, pages 523–533. Springer, 2017.

[45] O. J. A. Pinno, A. R. A. Grégio, and L. C. De Bona. Controlchain: Blockchain as a central enabler for access control authorizations in the IoT. In *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2017.

[46] M. H. Rasmussen et al. Recent changes in the building topology ontology. In *LDAC2017-5th Linked Data in Architecture and Construction Workshop*, 2017.

[47] T. L. Saaty. The analytic hierarchy process. *European Journal of Operational Research*, 48:9–26, 1990.

[48] P. Saint-Andre. Streaming XML with Jabber/XMPP. *IEEE Internet Computing*, 9(5):82–89, 2005.

[49] P. Saint-Andre et al. Extensible messaging and presence protocol (xmpp): Core. 2004.

[50] F. Salim, J. Reid, U. Dulleck, and E. Dawson. Budget-aware role based access control. *Computers & Security*, 35:37–50, 2013.

[51] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[52] L. Seitz, G. Selander, and C. Gehrmann. Authorization framework for the internet-of-things. In *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–6. IEEE, 2013.

[53] C. Shen and F. Pena-Mora. Blockchain for cities—a systematic literature review. *Ieee Access*, 6:76787–76819, 2018.

[54] N. Skandhakumar, J. Reid, E. Dawson, R. Drogemuller, and F. Salim. An authorization framework using building information models. *The Computer Journal*, 55(10):1244–1264, 2012.

[55] N. Skandhakumar, F. Salim, J. Reid, R. Drogemuller, and E. Dawson. Graph theory based representation of building information models for access control applications. *Automation in Construction*, 68:44–51, 2016.

[56] N. Skandhakumar, J. Reid, F. Salim, and E. Dawson. A policy model for access control using building information models. *International Journal of Critical Infrastructure Protection*, 23:1–10, 2018.

[57] B. Succar. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, 18(3):357–375, 2009.

[58] N. Szabo. Smart contracts. *Unpublished manuscript*, 1994.

[59] S. Taneja, B. Akinci, J. H. Garrett Jr, and L. Soibelman. Algorithms for automated generation of navigation models from building information models to support indoor map-matching. *Automation in Construction*, 61:24–41, 2016.

[60] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the ACM workshop on formal methods in security engineering*, pages 45–55. ACM, 2004.

[61] H. Watanabe et al. Blockchain contract: A complete consensus using blockchain. In *Proceedings of the 4th global conference on consumer electronics (GCCE)*, pages 577–578. IEEE, 2015.

[62] K. Zeilenga. Lightweight directory access protocol (ldap): Technical specification road map. Technical report, 2006.

[63] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan. Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal*, 6(2):1594–1605, 2018.

Appendices

A.1 Example Building

A.1.1 Building Graph Model

Below is a small sub-graph of the building's model (in RDF syntax) that can be queried using SPARQL as discussed in Section 4.2.3.

```
@prefix bf: <https://brickschema.org/schema/1.0.3/BrickFrame#> .
@prefix bot: <https://w3id.org/bot#> .
@prefix brick: <https://brickschema.org/schema/1.0.3/Brick#> .
@prefix building1: <http://building1.com#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

building1:AHU-1 a brick:AHU ;
    bf:feeds brick:VAV-1-12 .

building1:HVAC-Zone-1-12 a brick:HVAC ;
    bf:hasPart building1:Room-1-1-120,
        building1:Room-1-1-121 .

building1:Operations-Zone a bot:Zone ;
    bot:hasSpace building1:Room-1-1-120,
        building1:Room-1-1-121 .

building1:Room-B-100 bf:isLocationOf brick:AHU .

building1:VAV-1-12 a brick:VAV ;
    bf:feeds brick:HVAC-Zone-1-12 ;
    bf:hasPoint building1:Reheat-Command-1-12,
        building1:Temperature-Sensor-1-12,
        building1:Temperature-Setpoint-1-12 .

building1:Temperature-Sensor-1-12 a brick:Temperature_Sensor ;
    bf:controls building1:Reheat-Command-1-12 .

building1:Temperature-Setpoint-1-12 a brick:Temperature_Setpoint ;
    bf:controls building1:Reheat-Command-1-12 .

building1:Door-1-1-12 a bot:Element .

building1:Room-1-1-120 a brick:Room,
        bot:Space ;
    bot:adjacentElement building1:Door-1-1-12 .

building1:Room-1-1-121 a brick:Room,
        bot:Space ;
    bf:isLocationOf brick:Temperature-Sensor-1-12,
        brick:Temperature-Setpoint-1-12 ;
    bot:adjacentElement building1:Door-1-1-12 .

building1:Reheat-Command-1-12 a brick:Heating_Command .
```

A.1.2 Namespaces for Queries

```
PREFIX bf: <https://brickschema.org/schema/1.0.3/BrickFrame#> .
PREFIX building1: <http://building1.com#> .
PREFIX brick: <https://brickschema.org/schema/1.0.3/Brick#> .
PREFIX bot: <https://w3id.org/bot#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

A.1.3 Building Security Zones

The classification of security zones is carried out by the building manager based on the function of each room in the example building. Figure A.1.1 illustrates this classification.

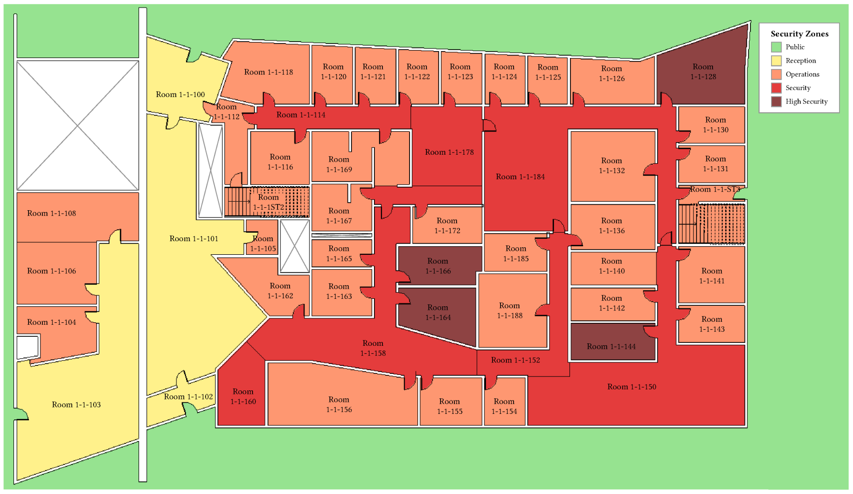


Figure A.1.1: The floor plan of our example building showing the classification of security zones

A.1.4 Example Pathways

Pathway 1

Below is the cost calculation for *Pathway 1* from Figure 4.1 using the cost function defined in Section 4.2.3.

$$\begin{aligned}\text{Pathway 1} \equiv \{&\text{Room\_1-1-102}, \text{Room\_1-1-101}, \text{Room\_1-1-100}, \text{Room\_1-1-112}, \text{Room\_1-1-114},\\ &\text{Room\_1-1-178}, \text{Room\_1-1-184}, \text{Room\_1-1-152}, \text{Room\_1-1-150}, \text{Room\_1-1-144}\}\end{aligned}$$

$$\begin{aligned}
\text{cost(Room\_1-1-102)} &= 1 + 0\\
\text{cost(Room\_1-1-101)} &= 1 + 0\\
\text{cost(Room\_1-1-100)} &= 1 + 0\\
\text{cost(Room\_1-1-112)} &= 3 + 0\\
\text{cost(Room\_1-1-114)} &= 3 + 0\\
\text{cost(Room\_1-1-178)} &= 3 + 0\\
\text{cost(Room\_1-1-184)} &= 3 + ((0.204 \times (1+5)) + (0.246 \times (1+5)) + (0.347 \times (1+5)) + (0.413 \times (1+5)))\\
\text{cost(Room\_1-1-152)} &= 3 + 0\\
\text{cost(Room\_1-1-150)} &= 3 + (0.413 \times (1+4))\\
\text{cost(Room\_1-1-144)} &= 4 + ((0.413 \times (1+20)) + (0.260 \times (1+20)) + (0.328 \times (1+20)))\\
\text{cost(Pathway 1)} &= 55.35
\end{aligned}$$

Pathway 2

Below is the cost calculation for *Pathway 2* from Figure 4.1 using the cost function defined in Section 4.2.3.

$$\text{Pathway 2} \equiv \{\text{Room\_1-1-1ST3}, \text{Room\_1-1-184}, \text{Room\_1-1-150}, \text{Room\_1-1-144}\}$$

$$\begin{aligned}
\text{cost(Room\_1-1-1ST3)} &= 2 + 0\\
\text{cost(Room\_1-1-184)} &= 3 + ((0.204 \times (1+5)) + (0.246 \times (1+5)) + (0.347 \times (1+5)) + (0.413 \times (1+5)))\\
\text{cost(Room\_1-1-150)} &= 3 + (0.413 \times (1+4))\\
\text{cost(Room\_1-1-144)} &= 4 + ((0.413 \times (1+20)) + (0.260 \times (1+20)) + (0.328 \times (1+20)))\\
\text{cost(Pathway 2)} &= 42.35
\end{aligned}$$

A.2 Analytic Hierarchy Process

A.2.1 Questions and Answers

We ask questions to compare the importance of different types of *Points*, e.g., *sensors* and *setpoints*. The *sensors* installed in our example building are temperature, damper position, humidity, and occupancy sensors. The *setpoints* available in our example building are temperature, humidity, and airflow setpoints. For each set of *Points*, we do a pairwise comparison of its elements using questions, such as ***How important/critical is element x compared to element y in the building?***. We answer these questions using the rating scale described in Appendix A.2.2. Each set of questions achieve a consistency ratio of less than 0.10 as recommended for the AHP process.

For the *sensors*, we achieve a consistency ratio of 0.024 answering the following questions:

- How important is Temperature Sensor compared to Damper Sensor? 2
- How important is Temperature Sensor compared to Occupancy Sensor? 1
- How important is Temperature Sensor compared to Humidity Sensor? 2
- How important is Damper Sensor compared to Occupancy Sensor? 1
- How important is Damper Sensor compared to Humidity Sensor? 1
- How important is Occupancy Sensor compared to Humidity Sensor? 1

For the *setpoints*, we achieve a consistency ratio of 0.055 answering the following questions:

- How important is Temperature Setpoint compared to Airflow Setpoint? 2
- How important is Temperature Setpoint compared to Humidity Setpoint? 1
- How important is Airflow Setpoint compared to Humidity Setpoint? 1

The resulting weights from answering these questions are shown in Table 4.2.

A.2.2 Rating Scale

Table A1: This rating scale is taken from [47] and used to compare two elements relative to each other in terms of their importance. Note that the reciprocal can also be used to compare elements in the other direction.

Rating	**Definition**
1	Equal importance
2	Equal to moderate importance of one over another
3	Moderate importance
4	Moderate to essential importance
5	Essential or strong importance
6	Essential to very strong importance
7	Very strong importance
8	Very strong to extreme importance
9	Extreme importance